Using Womb Grammars for Inducing the Grammar of a Subset of Yorùbá Noun Phrases

by

Ifeoluwanimi Adebara

M.Sc., University of Birmingham, 2012
B.Sc., University of Ibadan, 2009

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2017

Approval

Name:	**Ifeoluwanimi Adebara**
Degree:	**Master of Science (Computing Science)**
Title:	***Using Womb Grammars for Inducing the Grammar of a Subset of Yorùbá Noun Phrases***
Examining Committee:	**Chair:** Binay Bhattacharya Professor

Veronica Dahl
Senior Supervisor
Professor _____

Fred Popowich
Supervisor
Professor _____

Trude Heift
Internal Examiner
Professor
Department of Linguistics _____

Date Defended: 30 August 2016

Abstract

We address the problem of inducing the grammar of an under-resourced language, Yorùbá, from the grammar of English using an efficient and linguistically savvy, constraint solving model of grammar induction - Womb Grammars (WG). Our proposed methodology employs Womb Grammars in parsing a subset of noun phrases of the target language Yorùbá, from the grammar of the source language English, which is described as properties between pairs of constituents. Our model is implemented in CHRG (Constraint Handling Rule Grammars) and has been used for inducing the grammar of a useful subset of Yorùbá noun phrases. Interesting extensions to the original Womb Grammar model are presented, motivated by the specific needs of Yorùbá and similar tone languages.

Keywords: property grammars, constraint handling rules grammar, constraint handling rules, womb grammars, linguistics, Yorùbá.

Dedication

Agbanilágbàtán

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 The Problem Statement

Grammatical induction also known as grammatical inference has its roots in a variety of separate fields, and this explains why it has wide applicability. It is intuitively a difficult problem to solve and the precise difficulty of a particular inference problem is determined by the complexity of the target language, and the information available to the inference algorithm about the target language. Naturally, simpler languages and more information both lead to easier inference problems.

In the case of natural language, the difficulty of grammar induction is influenced by many factors. First, natural languages have a very large lexicon and new words are added so often, so that for instance, thousands of words are added to the lexicon of English every year. Second, the words in the lexicon can be combined together to form an infinite number of phrases, sentences and structures larger than the word. Also, because language is dynamic, new stuctures are constantly developed. Third, is the problem of ambiguity which results in the same word having different entries in the lexicon, having different meanings in different contexts or other semantic, syntactic or pragmatic ambiguity. These factors, and several others like the unrestrictive nature of language, language change through language contact, language borrowings, grammatical errors, code mixing and more, make grammatical inference in natural language a very hard problem.

Grammar induction ultimately aims at extracting meaning *(semantics)* from text. It is formal grammars that specify relationship between text parts of speech such as nouns, verbs, and adjectives and syntax: the implicit structure. However, we focus on efficiently inferring a linguistically accurate syntactic structure of input phrases, as well as syntactico-semantic relationships (such as features, see chapter 4.2) responsible for certain semantic structures and we explicitly output this relationship for conditional properties. All other semantics relationships analysis are beyond the scope of this work. Scientifically, we wanted to explore to what extent the parsing-as-constraint-solving paradigm of Natural Language Processing

problem solving could achieve a great degree of linguistic descriptive formality without sacrificing efficiency, in the realm of grammar induction and in particular for inducing Yorùbá, which is a severely under-resourced and endangered.

1.2 The Motivation

Language endangerment and death has been of serious concern in linguistics and language policy making. Close to seven thousand languages are currently spoken in the world, the majority of which are understudied and endangered. It has been said that an alarming 50 to 90 percent of languages will be extinct by the end of the century [79].

For various reasons, some speakers of many minor, less studied languages may learn to use a different language from their mother tongue and may even stop using their native languages. Parents may begin to use only that second language with their children and gradually the transmission of the native language to the next generation is reduced and may even cease. As a result, only the elderly in such communities may use the native language, after a while, there may be no speakers who use the language as their first or primary language and eventually the language may no longer be used at all. Thus, a language may become extinct, existing perhaps only in recordings, written records and transcription and languages which have not been adequately documented completely disappear.

Linguists cannot keep up with the study of these languages even for educational purposes, and there is a growing need for their automatic processing as well, since the amount of text sources grows much faster than humans can process them. To make matters worse, most linguistic resources are poured into English and a handful of other "first world" languages, leaving the vast majority of languages and dialects under-explored. Clearly, automating the discovery of an arbitrary language's grammar would render phenomenal service to the study and preservation of linguistic diversity.

1.3 Our Contribution

We adopt a novel grammar induction model, called, Womb Grammars (WGs), developed by Dahl and Miralles [43]. WGs employ a property based methodology, which, relies on the grammar of a source language to induce the grammar of a target language. WGs assume that the grammar of the source language is correct and that the lexicon and input phrases of the target language are correct and representative of phrases in the target language. It adopts a constraint satisfaction approach whereby input phrases of the target language are tested for satisfaction and unsatisfied constraints provide a lead for the reconstruction of the target grammar using the source grammar.

In this work, WGs are used to induce the grammar of a subset of noun phrases in Yorùbá, from that of an English grammar with same coverage [42]. We use English as

our source language and Yorùbá as the target language. English was chosen as the source language for various reasons, especially because, it is a language the author can speak and write fluently. English also has a wide audience, and, we believe that it will be easier for a wider audience to understand the WGs model using a language as source such as English, whose structure is well known. Using a well known language eliminates the need for the reader to in addition to understanding the WG model, learn and understand the structure of two less known languages (source and target languages). Also, because English is a language which has been well studied by linguists, many linguistic resources are available and easily accessible, and this made creating a correct property grammar (details in section 4.1) easier than using any language which has less resources easily accessible. The author's formal training in linguistics and native speaker competence in Yorùbá also made Yorùbá a desirable language as the target. Since we need a correct grammar of both the source and target language, English and Yorùbá languages were thus, the most accessible languages for this task. In fact, the success achieved from using English to induce the grammar of Yorùbá, reveals that, it is possible to use WGs for any pair of languages, including unrelated languages, although with some modifications.

The novel approach of WGs, needs neither a pre-specified model family, nor parallel corpora, nor any of the typical models of machine learning and works for more than just specific tasks such as disambiguation[43]. It automatically transforms a given (task-independent) grammar description of a source language, from just the lexicon of the target language and a representative input set of correct phrases in the target language.

The syntactic descriptions of the source and the target language subset addressed are stated in terms of linguistic constraints (also called "properties" in linguistic literature) between pairs of constituents, although for the target language constraints we depart from the classic formalism [23] in view of Yorùbá motivated extensions. This allows us a great degree of modularity, in which constraints can be checked efficiently through constraint solving.

This work differs from the previous model[43] in the following ways:

1 A failure driven approach was adopted in previous models, however we use a failure conscious and success conscious approach (details in chapter 5.2.1).

2 Previously, WGs tested each property for failure only once in the entire input phrases, however, due to the peculiarities of Yorùbá, we test each property for failure and success in each input phrase (details in chapter 5.2.1).

3 While the grammar of the target language is implied from the level of satisfaction of the properties tested, we explicitly reconstruct the grammar of the target language from that of the source language(details in 5.3).

4 We induce conditional properties that involve the features of each word of the target language (details in chapter 5.3.7).

5 We provide a count of the number of times each property fails and / or succeeds. This is to enable users to have a deeper insight to how often or otherwise a property fails and / or succeeds in a language (more in 5.2.2).

6 The previous version used only two traits (which we refer to as features) in describing each word. We, however, introduce four additional features including tone. This makes this version adaptable to other languages including tone languages (details in chapter 4.2).

7 We provide a record of English gloss for every word in the target phrase (details in chapter 5.2.4).

8 Previously, only properties present in the source language can be inferred for the target language. However, we extend this model to infer some properties which are present in the target language but absent in the source (details in chapter 5.2.6). Morphologically rich languages like Russian, may require an additional module to handle the morphology of the language.

9 We explicitly output each failed and/or succeeded property, the word boundaries and phrase boundaries for each input phrase (details in chapter 5.2.5)

10 Gender and number are syntactically defined in Yorùbá and as a result, we explicitly update the gender and number of certain words in context according to Yorùbá rules (more in chapters 3.5, 5.2.7).

11 While previous models of WGs do not test input phrases for constituency, this model explictly tests the constituency property (details in chapter 5). We explicitly test for constituency to ensure our system does not add the constituents in the source lanuage that are absent in the target language.

12 We use a context-free grammar of the target language to evaluate the correctness of our model.

Our results in applying and adapting the WG model for inducing Yorùbá noun phrases show that this model compares favourably with others in solving the grammar induction problem: it combines linguistic formality with efficient problem solving, and can transfer into other languages, including languages in which tones have a grammatical and / or semantic function.

Chapter 2

Literature Review: Grammar Induction

There are multiple levels of language analysis and [39] divides the structure of human language into *Phonology, Morphology, Syntax, Semantics* and *Pragmatics*. The multiple levels of language interact beginning in the case of spoken language with the sounds produced (phonology) to the meanings (semantics, pragmatics) that those sounds express.

Phonetics delves into the realm of speech sounds also referred to as *phones* and is fundamental to the entire field of liguistics. Phonology studies the processes responsible for phone patterns in a language. It focuses on groups of phones that are equivalent for a given language (allophones), groups of sounds that account for meaning distinction in a language (phonemes) and processes such as for instance assimilation which determine these variations. An example is, the plural marker *s* which is realized as *z* in *boys* and as *s* in *books* because of an assimilation process that is responsible for the plural marker being realized as *z* after voiced sounds or vowels and as *s* after voiceless sounds.

Morphology focuses on word formation. Phonemes in turn are the constituents of morphemes (minimal meaningful word segments) and these are building blocks for words (words are a combination of one or more morphemes). Word formation processes using inflections, derivations, compounding, affixes, reduplications and more are studied in the field of morphology. Syntax studies how words are grouped into phrases and / or clauses, such as noun phrases, verb phrases, adverb phrases adjective phrases and prepositional phrases. It is a set of rules for arranging words into more complex units of language (sentences) and which arrangment is grammatically correct in a language. Theories like universal grammar, transformational generative grammars, syntactic trees and other syntactic compositions are studied in the field of syntax.

At still higher levels of semantics and pragmatics, the focus is on linguistic meaning conveyed through language. While semantics focuses on lexical (morpheme and word meaning) and compositional meaning (phrasal and sentence meaning), pragmatics is concerned with

meaning of language in context. In semantics, the meaning of morphemes, words and phrases combine to provide meaning for the sentence [39].

Techniques have been developed for language analysis at all of these multiple levels. However, we will focus on syntax which has been studied by many researchers.

2.1 Grammar Induction

2.1.1 A general overview of grammar induction

Grammar induction "is a method of inquiry that tries to understand the computations involved in making inferences about grammars from observations under a variety of different learning scenarios." [58]. The "grammar" in "grammar induction" are models of knowledge representation that may include the formalisms, logics, rules, constraints, and structures that compose the systems underlying natural language. "Induction" refers to the process involved in acquiring knowledge through hypotheses and observations [58].

"A grammar **G** is a finite representation that describes a (possibly infinite) language G_L" [58]. A language from a formal perspective is a set of finite or infinite strings, and the corresponding language is called a finite or infinite language respectively. A grammar thus, represents a language and the grammar of a language can be viewed as a theory of the structure of the language. Also, a grammar is based on a finite number of observed sentences which may also be referred to as a corpus, and this is projected to an infinite set of grammatical sentences. This is achieved by establishing general grammatical rules which are framed in terms of hypothetical constructs (phonemes, morphemes, etc) of the language under analysis. A properly formulated grammar therefore should determine the set of grammatical sentences of a language unambiguously [33]. Grammar has been categorised into prescriptive, descriptive, formal and generative grammars [34].

Prescriptive grammars prescribe authoritative norms for a language regarding how people should talk and write and / or how not to talk and write. The prescriptive grammar thus serves to mold the spoken and written language of a particular class of people to some acceptable standard or norm. For instance, in English, a noun and a verb is expected to agree in number, so that *John sings* is considered grammatical, and correct while for *John sing* the opposite is the case. These kinds of rules are prescriptive grammar rules of a language [34].

Descriptive grammars attempt to describe actual usage of a language rather than enforce rules arbitrarily. Studying the properties of languages and clarifying the basic conceptions that underlie them is useful for arriving at a theory of linguistic structure. Scientific theories are based on a certain finite set of observations and, by establishing general laws stated in terms of certain hypothetical constructs, it attempts to account for these observations, to show how they are interrelated, and to predict an indefinite number of new phenomena. The

first step in the linguistic analysis of a language therefore, is to provide a finite system of representation for its sentences based on certain finite set of observations [33]. A descriptive grammar of English for instance will say that determiners precede nouns and nouns are compulsory constituents of English noun phrase. This kind of grammar only says what is discovered through observations of a finite set of expression otherwise known as a corpus.

A *formal grammar* is a precise symbolic representation of a grammar, usually specified by a system of rules and they can be used to either generate grammatical sentences or parse sentences in order to discover their grammatical structure. The grammar a formal grammar represents can either be descriptive or prescriptive in nature. A formal grammar is a 4 tuple G = (N, Σ, P, S) such that:

1. N is a finite set of non-terminals

2. Σ, is a finite set of terminals and is disjoint from N

3. P, is a finite set of production rules

4. S $\in$ N, is the start symbol

Formal grammars are usually represented with grammatical symbols in order to avoid ambiguity. Chomsky [34] identified four hierachies for formal grammars: free or unrestricted grammars, context-sensitive grammars, context-free grammars and regular grammars [58]. These hierarchies of grammar are representations of different types languages.

1 Free or unrestricted grammars, also called type zero grammars or recursively enumerable grammars are grammars accepted by Turing machines (more details in appendix D). Logic grammars are examples of type zero grammars [2]

2 Context-sensitive grammars are the second level of the hierarchy and they are accepted by linear bounded automata. Their rules are of the form: αAβ, $\rightarrow$ αBβ; S $\rightarrow$ ϵ, Where A, S $\in$ N; α, β, B $\in$ (N $\cup$ Σ)$*$; B $\neq$ ϵ. Grammars of some natural languages, e.g germanic languages are context sensitive (more details in appendix D).

3 The third hierarchy, context-free grammars are grammars in which the left-hand side of each production rule consists of only a single nonterminal symbol. Grammars of many natural languages can be defined with context-free formalisms. For example, the context-free rule $NP \rightarrow D, N$ is a rule that rewrites any occurrence of *NP (noun phrase)* into an occurrence of a *D (determiner)* and a *N (noun)* in English noun phrases (more details in appendix D).

4 Regular grammars are those where the left hand side is only a single nonterminal symbol, and the right-hand side is also restricted. The right side may be the empty string, a single terminal symbol, or a single terminal symbol followed by a nonterminal symbol (more details in appendix D).

Generative grammars are a variety of formal grammars borne out of the view that humans have an innate *language faculty* which is reflected in how sentences are generated by a subconscious set of procedures. These procedures are part of our cognitive abilities so that when children learn their native languages, they acquire specific rules that determine their choice of words, how words are strung together to form phrases, sentences etc as well as the meanings these expressions convey. These rules are learned in a relatively short time with little or no conscious effort from the children and these rules interact with each other in complex ways. The goal of generative grammar is to model these procedures in order to generate the grammar of a language. In other words, finding a procedure to figure out what humans subconsciously know about the grammar of a language. In generative grammar, the means for modeling these procedures is through a set of formal grammatical rules [34, 33]. In English, for example, information such as: the subject of a sentence precedes a verb, can be encoded as a generative rule using a formal grammar symbolism. These rules are referred to as generative grammars because they are thought to generate the sentences of a language. Tree adjoining grammars, affix grammars and attributive grammars are some examples of generative grammars [60].

Grammar induction is influenced by formal language theory which studies how knowledge regarding a natural language can be modeled with a formal language. For instance, a formal language is useful for modelling the knowledge of which sentences in a natural language are well formed such that the formal language contains all and only those sequences of well formed sentences [58]. In phonology for example, there are some sequences of phonemes (words) which a native speaker of any language would recognize as not being a part of their language. This is not because they know all the words of their language, but because they have the innate knowledge of well-formed combinations of sounds, and this can be modeled with a formal language that comprises all and only those sequences of phonemes which make up the plausible well-formed words of the language.

In order to represent a finite [1] language, one can simply return a list of all strings in the set. This however becomes impossible, if we try to do the same for an infinite [2] language. So that, instead of trying to return the (infinite) list of all strings in the set, we can resort to grammatical devices such as recursive rewriting rules for example $Sentence \rightarrow Nounphrase, Verbphrase$, that allow us to describe an infinite set of sentences with a finite set of grammar rules. In order to represent an infinite set with finite means therefore, there arises the need for a grammar to provide a finite description of ways of combining words into sentences, like our recursive rewriting rule above. This grammar is denoted by a finite representation of the words and sentences of these formal languages, thus making it possible to accommodate any plausible combination of well-formed strings of the language [58]. For example, property grammars introduced by [23, 22] is a grammar

[1] A finite language is one that contains a finite number of strings.

[2] An infinite language is one that consists of an infinite number of strings.

representation formalism for expressing a possibly infinite set of sentences through a finite set of property descriptions such as precedence, requirement, uniqueness, etc, properties of language and can accommodate any plausible combination of 'well-formed' categories of a language. In essence, to describe languages, which may be finite or infinite, we require predefined grammars (or we need to define them ourselves), and each grammar represents its own finite or infinite language.

The grammatical inference process involves extracting language information from a language model by means of an oracle [3]. The language model can be a description, representation, or model of the correct phrases, sentences or any structure of the language. The oracle can be a human expert, also called an informant, who provides information about the language, the world wide web, a treebank or anything that can provide the required information. The oracle provides the learner or grammatical induction model with information which typically consists of strings according to the model, and it accesses the information from a grammar, description, representation, or language model available to it. The learner uses the aforementioned information to create its own model of the language, also known as the induced language and / or grammar [58].

The grammar induction model usually has access to some language data which may be sequential [67] or structured (graphs, words, trees etc) and is asked to return a grammar of the language to be learned (we will refer to the language to be learned as target language henceforth) which consists of a model of the target language and includes an explanation or a labelling of the input data. The explanation or labels can be parts of speech tags, a label of whether an expression is well-formed or not, or any other label or description desired. For instance, our womb grammar model has access to data: correct phrases and a lexicon of the target language, and returns a grammatical description of the structure of input phrases in the target language, described in the property grammar formalism.

[3] An oracle is an entity in the learning environment (if a parent, I would not call him or her a "learning environment") that provides information from the language that is to be learned.

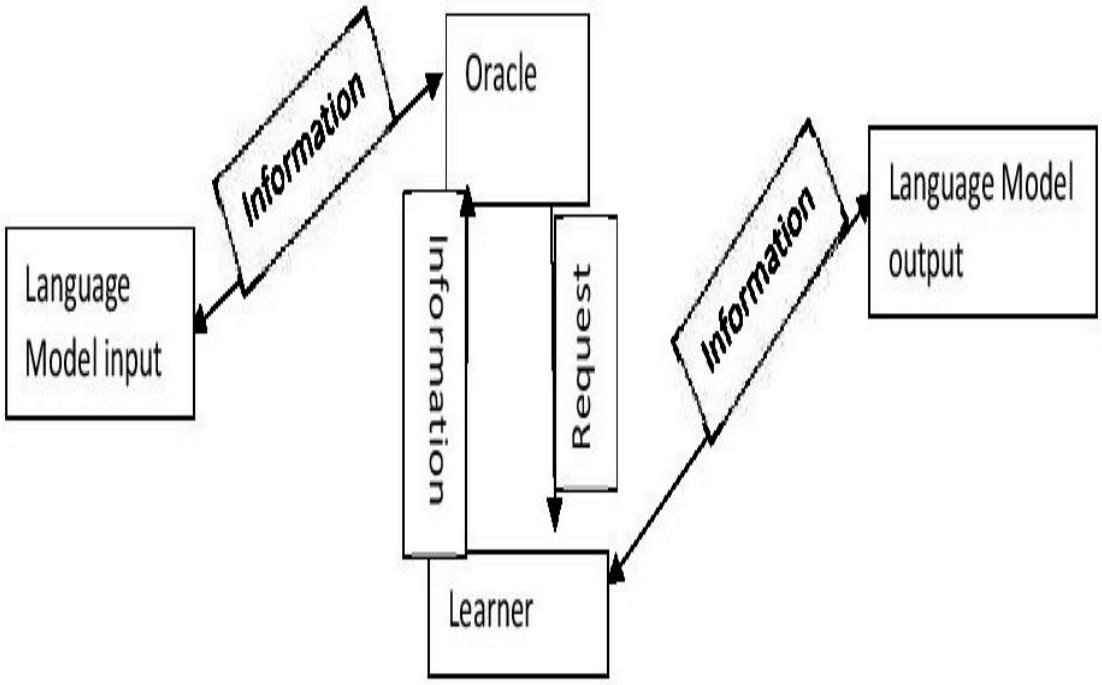


Figure 2.1: A General Process of Grammar induction

From figure 2.1, for example, in WGs, the oracle is a human expert who provides example noun phrases and words in the lexicon, and these are used as input for the WG system. The examples received from the oracle is the language model input as in figure 2.1, the oracle is the human expert, the learner is the constraints in the WG model and the language model output is the property grammar output of the target phrases (more details in chapter 6).

Formal and empirical grammar induction have been mentioned in [58]. The formal approach is concerned with whether specific families of languages as a whole can be efficiently identified under certain conditions. The results are shown by providing formal, mathematical proofs of learnability. Empirical induction is however concerned with grammatical inference of specific individual languages. The results are a specification or approximation of the grammar of the language. We will focus on empirical approaches.

2.1.2 The different paradigms for grammar induction

Speed, convergence of learning, type of input data (labelled or unlabelled) and quantity of resources needed to learn are some criteria usually imposed on the language data in order to secure a learning result. For instance, the criteria for a specific grammar induction task may be to ensure that the data is representative of the target language, or to use correct samples from the target language, or even to use a combination of both correct and incorrect samples, etc. The results of a grammar induction task can be measured

as a specific number of iterations, a convergence to a predetermined good solution, etc, and these criteria are used in different learning scenarios(active learning, probablistic, or non-probablistic scenarios, etc). For instance, our womb grammar model requires that only correct words and phrases of the target language are used for induction. The learning result is also secured when all phrases have been tested with all relevant constraints (more of this in chapter 6). In a non-probablistic setting [53], the criteria is such that the learner has access to a "never ending" supply of information about the target language. These could be actual strings of the language or labelled strings, each label indicating whether or not the string belongs to the target language. Positive samples are made of strings or sequences over an alphabet which are in the target language and negative samples are made of a set of strings or sequences over an alphabet not belonging to the target language [32, 48, 86, 46]. The probablistic setting however extends the criteria of the non-probablistic setting to include probabilities for learning distribution [58].

In another setting called *active learning* [15], no data is directly available at first, or, only unlabeled data is available. The inductive model is therefore expected to induce or learn, as well as query the environment in order to obtain its data and / or a labeling of its data. Membership queries and equivalence queries are two important queries in this setting. The learner tries to find out if a certain string belongs to the language or not by asking an oracle questions in what is called a membership query. In equivalence queries, the learner can present the oracle with a candidate grammar and is expected to either answer yes, or provide the learner with a counter-example [58]. Active learning uses a technique called selective sampling to select the most informative examples, from a pool of unlabeled data and the learner is able to determine the most informative samples through a predetermined criteria. Selective sampling is usually achieved using two major methods: committee-based sampling and certainty based sampling [15]. While the former relies on a group of learners which have different hypotheses regarding the learning task, the later are those which select samples which a single learner is least confident of, as determined by some selection criterion.

Active learning reduces the amount of human-annotated training data needed to train models and it has been relevant for learning low-resource languages which do not have large corpuses [90]. An active learning process thus involves a number of examples, batch size, supervised learner, data set and a stopping criterion. The supervised learner is trained with the input examples; the batch size is the number of examples that are queried or sampled from the unlabeled pool, and added to the training pool at every iteration. The stopping criterion is the condition that ensures the learning process ends at some point. It can be either a desired performance level or the number of iterations. The iterative process of training, selecting samples and annotation is thus repeated until there's a convergence or after a number of iterations. Evaluation is usually by comparing the learner's performance with a traditional parser's. The active learner is said to have a better performance than the traditional parser if the number of labelled data used to achieve the performance level

is lower than that used for the traditional parser. The performance level is also determined by comparing the results with those of the test set at every iteration [58].

Learning settings like *online learning, interactive learning*, and *batch learning* have also been explored [58] and the learner can access data either as samples, via access to a simulator [15], or by running experiments. The learning model which can also be called a learning algorithm, induction model or an inference model is at least partially automatic and the model can then be used to compress, classify or provide a model for input data. In the case of language, input data can be words, phrases, sentences or any other language structure and the inference model is used to provide the grammatical information for instance, the syntactic information the data conveys.

2.1.3 Supervised, unsupervised and semi-supervised models

A grammar induction paradigm based on the type of information its input contains can belong to three classes: supervised, unsupervised and semi-supervised models and this grouping is based on the input data type [58]. In *supervised learning*, the input data is usually annotated manually by a qualified professional and the human expert would label the set of instances that will be used for the induction process. The process of labelling the data is a tedious and expensive process which requires high level skill and attention to detail in order to avoid errors. The more labeled instances the induction model has access to, the more precise the output, so that the amount of annotated data needed to reach a desired quality is time-consuming. The advantage of using hand labelled inputs is that the system can be better controlled: the system for instance, can be made more efficient by ensuring that only non-redundant / relevant data, is available during the learning process. For example, a system designed to induce the structure of noun phrases can be made to have access to only noun phrase examples [59, 53].

Unsupervised learning does not use labeled data but attempts to find out similar patterns in the data to determine the output. This approach was motivated because of the scarcity of annotated data and the cost of producing one. *Semi-supervised learning* is a method that uses both labelled and unlabelled data and the labeled data is used to help with labeling the unlabeled data.

2.1.4 Formalisms of Grammar induction and results

Several formalisms are being used to describe natural languages and many grammar induction algorithms have been developed to induce *deterministic finite automaton* [30], *context-free grammars* [31], *tree automaton* [74], *categorial grammars*, *stochastic automata* [30], *stochastic context-free grammars* [63], *probablistic grammars* [92], *property grammars* [7, 4, 3] and more. These algorithms have been differently classified according to the various features of the induction process.[13] provided a classification based on the underlying learn-

ing paradigm into: models based on Categorial Grammar, memory-based learning models, evolutionary computing models, and string-pattern searches. In [49], grammar induction methods are classified according to the type of input, which have been split into methods that learn from a tagged dataset and methods that learn from untagged / raw data of text. Classification as tag-based and word-based methods are also described [40], [46].

Categorial grammars are a lexicalized theory of grammar that does not include a separate collection of rules for combining words. Rather, lexical categories of words such as nouns and adjectives constitute functions that determine how these words can combine with other categories [56]. They are said to be a foundation for many *unsupervised models* mainly because the lexicon and the grammar are induced in the same task. It has been said to be useful for producing a realistic psychological model of human language acquisition and also for simpler and more efficient algorithms for the complex task of grammar induction [13]. Memory-based models are those used to abstract information from a given data whilst retaining all information and maintaining accuracy. It is useful in the analysis of unseen data and its good performance is due to the fact that it does not forget any training examples. Evolutionary computing models on the other hand use techniques similar to the evolution process that humans use to learn, understand and communicate using language to develope language acquisition devices. This approach though complicated is useful for allowing natural selection which eventually leads to optimal language learning conditions. All other algorithms which do not fit into any of the afore mentioned paradigms are captured under string-pattern searches [13].

Grammatical induction has been extensively addressed by researchers in various fields. In automata theory, finite-state automata has been used to decide whether a string belongs to a certain language or not. A finite state automata accepts regular languages and it is a 5 tuple, A = (Σ, Q, q0, F, δ) such that:

1. Σ, is the input alphabet

2. Q, is the finite set of state

3. q0 $\in$ Q, the start or initial state

4. F $\subseteq$ Q be the set of final states

5. δ is a transition function. This function's arguments typically takes a state and input symbol and returns a state. For instance, let p and q be states, and b be an input symbol. A rule δ(p,b) = q means that a state p, with an input symbol b transitions to state q.

A finite-state automaton thus, is made of states as well as transitions between these states, which are labeled by symbols from the predetermined input, also known as alphabet. At least one state is chosen as the initial state, and some states are marked as accepting or

end states. An automaton is able to recognize a string if there exists a series of transitions from an initial state to an end state that reads the string. The language an automaton recognizes, is equivalent to the set of string it recognizes [58].

In language acquisition, grammatical inference has been explored in relation to the role of semantics in how children acquire language [14], it has also been used for developing models for empirical analysis of first language acquisition [36] and to answer several other language acquisition questions [11, 36, 12]. In the field of computational linguistics, the focus of grammatical inference has been on how to automate the processes and decisions made during communication with human language [58]. Several algorithms have been developed for various aspects of human language communication such as algorithms to capture and use acoustic-phonetic variation during speech recognition [71], to determine the boundary of words in continuous speech [77], that learn phonotactic [4] grammars of natural language phonology [92], for morphological analysis [57], and for solving several other natural language related problems [41, 25, 55] with the aim of multilingual translation [78].

Machine learning approaches [53] have also been used for grammar induction tasks. A corpus of examples is divided into a training set and a test set. The learning algorithm and a model of the task to be learned are specified. The algorithm then learns from the training data to construct a parser that assigns syntactic annotations to input strings from the test data. A gold standard of correct parses in a corpus may then be used to evaluate the model by computing a percentage of correct parses that the algorithm produces when tested on the test corpus. Machine learning algorithms are also more commonly evaluated for performance by scoring the models ability to recall information and also for precision. In evaluating an algorithm for recall, a percentage of structures the algorithm is able to correctly identify in a given set is calculated. The precision on the other is measured as the percentage of structures (recalled) which correspond to those in the gold standard. An algorithm's f-score is computed as an average of its recall and its precision.

Grammar induction has also been useful for using linguistic information from one language to describe another language and it has yielded good results. However, it was used for specific tasks like disambiguating the other language [29], or fixing morphological or syntactic differences by modifying tree-based rules [68], rather than for syntax induction which is our focus. This approach usually requires parallel corpora, although, there exists an exception[38] where a bilingual dataset is used to train parsers for two languages simultaneously. This is accomplished by adding grammar weights in the two hidden grammars, and is useful for learning dependency structure in an unsupervised empirical Bayesian framework.

[4] the study of the rules governing the possible phoneme sequences in a language

2.1.5 Evaluation of grammar induction results

The aim of grammar induction models is to learn the grammar of the language which performs best for any given task or group of tasks, making it an optimization problem, where the most efficient and optimal solution is preferred. In order to be able to determine how well a model performs, as well as compare it with other models, there is need for an evaluation criteria to be specified. Several evaluation approaches have been discussed in the literature. Some of them are: rebuilding well known grammars, looks-good-to-me approach, comparing against a tree bank, etc and each have their advantages and disadvantages[58].

In the evaluation criteria that involves rebuilding well known grammars, one or more toy grammars are selected. These grammars, which can be a context-free grammar or any other, are used to generate data that will serve as input for the grammar induction model. The output of the grammar induction model is then compared with the original grammar to find language or structure equivalence. This evaluation method eliminates the need for sequential data and additional data can be generated easily, if there is a need during the induction process. However, it can allow for unfair comparisons if the grammar is selected specifically for the algorithm [58].

The looks-good-to-me approach is applied to a collection of sequential data. The output of the system, which can either be in the form of a treebank of the input data or a grammar, is then evaluated manually. The evaluation may be a visual inspection of specific constructions such as certain grammar rules, whether recursions occur, certain peculiar language structures or tree structures. If the expected structures are found, the system will be considered to be performing well. This approach is easy to use and realistic because only unstructured data is required, which is easily accessible for several languages. The main drawback of this approach however, is the bias of the evaluation expert, who may be the developer of the system [58, 65]. Using an independent expert will often be a lot more useful. The evaluation is usually described in textual comments thus making it sometimes ambiguous and it varies from one evaluator to another [58].

Treebanks are seen as a gold standard and are considered to be correct. This is because they are developed based on a language theory and they thus serve as good evaluation models. In order to evaluate an inference model, plain strings which have been stripped of their annotations are input into the grammar induction model. The only structure the strings have will be the order in which words occur. The output of the grammar induction model is then compared with the gold standard treebank. A parser may be used to analyse the equivalence of the output with the gold standard if the output is not a tree structure. Precision and recall are used to determine how closely related two tree banks are [58]. In evaluating a model for recall, a percentage of structures the algorithm is able to correctly identify in a given set is calculated. The precision on the other is measured as the percentage of structures (recalled) which correspond to those in the gold standard. An algorithm's f-

score is computed as an average of its recall and its precision. This evaluation method is very objective, however, because many languages do not have treebanks, using this evaluation method is out of reach for grammar induction of many languages including Yorùbá.

2.1.6 Womb grammars for grammar induction

The Womb Grammar model is a novel approach that induces the grammar of a target language from a source language which does not even need to be related to the target. Like active learning models, WGs is able to perfom well even with little data, thus making it very useful for learning low-resourced languages. The approach is very straight forward and only requires a correct property grammar of the source language and positive data which consists of a lexicon of the target language, and a set of representative input phrases in the target language. This representative input phrases must include at least one unique example of each occurrence, i.e., grammatical structure present in the language or subset of language considered. WGs model abhors negative data because it is implemented under the premise of only correct input phrases and grammar. As a result, we used expert human informats (linguists and native speakers) as our oracle, due to the resource scarcity of Yorùbá. They provided us with correct noun phrases, each having different structures. Each word in these phrases was labelled, and the labelled words, together with the input phrases were used by WGs model to induce the grammar. Unlike machine learning algorithms which require training and testing data, our model needs only test the input phrases for constraint satisfaction and unsatisfation (as the case may be, more details in chapter 6), the results of which provides adequate clues for grammar induction. Using this formalism allows constraints to be evaluated independently of one another. The result of a parsed input string is either satisfied or unsatisfied and this allows flexibility in the representation of partial information, making it possible to describe any kind of output. The interest of such a conception is that every property in the grammar states some constraint on well formedness. It can be the case that all constraints are satisfied, but this is not an imperative condition, so that all kinds of input can consequently receive a characterization. Evaluation has been by comparing the results of our grammar induction with the linguistic descriptions of Yorùbá noun phrase described by linguists and available in literature. We also compared our results with a context-free grammar of Yorùbá noun phrases.

2.1.7 Open questions

In recent years, there have been advances in grammar induction of context-free grammars. Due to the complexity of natural languages and the high amount of resources necessary for producing induced grammars, research in automatic grammar induction continues to be of interest. This is especially true for developing unsupervised grammar inference models that

are also very accurate when compared to supervised or semi-supervised models. Models which are able to infer grammar from incomplete data, noisy data are also in demand [59].

2.2 Language documentation and description

2.2.1 A general overview of language documentation and description

Language documentation provides a record of the linguistic activity of a speech community, such as a collection of speech recordings which are subsequently transcribed as texts. Language description, on the other hand, presents a description of observed linguistic practices such as a description of a language's grammar, a phonological analysis of the sounds of the language, lexicography and other linguistic abstractions [83, 54].

Language documentation and description are fundamental to linguistics and have been beneficial in other areas of linguistics. Historical linguistics for instance has benefitted greatly from ancient texts, grammars and dictionaries in reconstructing a language's prehistory, classifying language families, etc. It has also been beneficial in developing theories related to language change and variation and also for testing theories of language typology and universal grammar [54].

Although language documentation and description are old subjects of research, they have experienced radical transformations due to the development of technologies for description, archiving, annotation, and dissemination of lingustic information. The new focus of language documentation otherwise known as documentary linguistics, which is the branch of linguistics that is *concerned with the methods, tools, and theoretical underpinnings for compiling a representative and lasting multipurpose record of a natural language or one of its varieties* [54, 16]. It has [16, 93] emerged as a result of the increasing need to maintain language diversity and document the world's languages not just for academic purposes but also to mitigate language endangerment and language death. The data gathered through language documentation are of tremendous benefit to grammatical induction.

Scholars such as [70] have argued for the separation of language documentation from description, because the methods and end results of language documentation and description are different. While the former collects primary data through interactions with native speakers, the later uses primary data to develop secondary data, which is a lingistic description of the primary data. However, because documentation is a means to an end, that is, the collection and archiving of data for consequent linguistic analysis or description, language description is dependent on language documentation and is seen as a complement to the documentation process [69].

In fact, in practice, the documentation process can rarely occur all at once but involves the collection of data and the analysis of data collected. This will eventually lead to the need to collect more data, clarify transcriptions and verify the reliability of data collected and / or

transcribed. Therefore, language documentation must rely on the application of descriptive linguistic methods in order to asertain the usability of data collected as well as to ensure that they are correctly transcribed and comprehensive. It is only through linguistic analysis that we can discover that some crucial speech genre, lexical form, grammatical paradigm or sentence construction is missing or under-represented in the documentary record. Without good analysis, recorded audio and video materials do not serve as data for any community of potential users. Similarly, linguistic description without documentary support is unverifiable.

2.2.2 Recent advances in documentary linguistics

Descriptive linguistics has experienced high level computational approaches to meet the ever increasing corpora now available for many languages. Over the last two decades, several tools have been developed with specialized support for various aspects of language documentation and description. In order to do their specialized linguistic processing, each of these tools depends on some model of linguistic information. Tools have been developed for data management, speech analysis and phonetics, phonology and morphology, syntax and grammar description, lexicon, text analysis, language survery and comparison.

Data management tools like the linguist's shoebox, SIL Fieldworks, LinguaLinks, Hypercard and many more are mainly used for managing lexicon, texts, and grammar [84, 26]. Speech analysis and phonetic tools like SpeechAnalyzer, Signalyze, FindPhone, Praat etc have been useful for acoustic and distributional analysis of speech sounds. In phonology and morphology, tools like AMPLE, PC-PATR, TonePars, PC-KIMMO have been useful in parsing. PC-PATR has also been useful for developing a syntactic corpus and grammar description, as well as Rook, a system for authoring descriptive grammars. IT, Concordance, Micro-OCP, shoebox, etc are useful for text analysis and interlinear annotated texts. WORDSURV for language survey and comparision and many other software tools are being developed to aid linguists in language documentation and description [84, 26]. However, no tool has been developed to induce the syntactic structure of a target language from a source grammar like our Womb grammar model does.

Documentary linguistics has also witnessed the initiative *Dokumentation Bedrohter Sprachen (DoBes)* funded by the Volkswagen foundation, Germany. The *Endangered Languages Documentation Programme (ELDP)*, *Electronic Meta-structure for Endangered Languages Documentation (EMELD)* and *Documenting Endangered Languages (DEL)* are some of the advancement trends in the field [26]. These initiatives have been developed in order to document languages that are potentially in danger of becoming extinct within a few years. They provide revitalization and maintenance for endangered languages and collect information regarding language diversity, cultural treasures and practices with the aim of preserving a communities linguistic and social culture. They also ensure that linguistic information is collected properly and that such information is submitted to documentation depositories.

The depositories contain music recordings and videos of cultural events and activities, pictures of cultural activities, audio and video recordings with annotations of differing depths: usually a transcription and a translation into one or more major languages is present and often morphosyntactic glossing is included as well. Documents on language's genetic affiliation, socio-linguistic context, phonetic and grammatical features, and the circumstances of research, recording and documentation are also stored in depositories.

Although the theoretical distinction between language documentation and descriptive linguistics is acknowledged, current linguistic trends [93], advocate a documentary process which leads to a comprehensive reference grammar and corpus of texts that can be used by linguists and speakers for a variety of purposes and our model provides a step in that direction. Especially because our model still produces intelligible results for incomplete data or data marked with errors like spelling mistakes and other grammatical errors introduced during writing or word processing. Our model is also able to provide comparisons for unrelated languages unlike other comparative syntax tools which are used for comparisons between closely related languages.

Chapter 3

The Yorùbá Language

Yorùbá belongs to the Yoruboid group of the Kwa branch of the Niger-Congo language family, which cuts across most of sub-Saharan Africa. It is a tonal dialect-continnum comprising about 20 distinctive dialects and spoken by over 30 million people in the western part of Nigeria [51]. Niger-Congo is the largest of the five main language families of Africa. The others being Nilo-Saharan, Afro-Asiatic, Khoisan and Austronesian (mainly found in the nation of Madagascar).

Yorùbá is one of the three regional (national language contained in the constitution) languages in Nigeria and is said to be the most studied African language. So that Yorùbá is a regional language in the south western part of Nigeria. Yorùbá is spoken by more than 20 percent of the population of Nigeria (the largest single black nation on earth), a country with a population of about 170 million people. The two other national languages are Hausa and Igbo, both of which are also regional languages in the north and southeastern parts of the country respectively. The Yorùbá language is a koine [51] a process involving dialect mixing, levelling, and simplification [89]. *Standard* Yorùbá which can also be referred to as the standard koine is a compendium of several dialects, mainly dialects of Ọyọ́, Ìbàdàn, Abẹ́òkúta and Lagos, which were major trade centers of early CMS missionary activities.

Apart from the standardized koine, there are twenty other dialects of Yorùbá: Ọyọ́, Ìjẹ̀ṣà, Ìlá, Ìjẹ̀bú, Onḿdó, Wo, Owe, Jumu, Iworro, Igbonna, Yagba, Gbedde, Ẹ̀gbá, Akono, Aworo, Bunu (Bini), Èkìtì, Ìlàjẹ, Ìkálẹ̀, Awori. These dialects are largely mutually intelligible, albeit with some variations in vocabulary and phonology and were largely spoken by different groups of people who, though tracing their descent to their common progenitor (Oduduwa), did not consider themselves as one people. Although there are several dialects of Yorùbá, it is important to mention that the present research is based on the 'standard' Yorùbá. This is because, 'standard' Yorùbá is the only variety referred to as Yorùbá and the only variety taught in schools in Yorùbá land and diaspora. It is the variety used whenever media uses Yorùbá and the variety used as a regional language. It is described as a resource scarce language [9] due to the limited number of texts available in Yorùbá.

3.1 The Sociolinguistic Situation of Yorùbá in Nigeria

Despite the seemingly large population of Yorùbá speakers, according to [21], it is a deprived language. A deprived language is one which although taught in schools, is dominated, as Yorùbá is, by a powerful language, such as English. As we will explain later, many schools do not use the Yorùbá at all, but will rather use English. Yorùbá has also been classified as a seriously endangered language [50], based on [94], who categorised language status into:

(i) potentially endangered - there is a decline in the use of the language by children

(ii) endangered - youths have experienced a decline in the use of the language and children experience language loss

(iii) seriously endangered - competent speakers are about 50 years old and above

(iv) moribund - competent speakers are very old, and many are dead

(v) extinct - when the last speaker dies.

This is influenced by the language policy of Nigeria which favours the use of English above all indigenous languages. It also pays lip service to for instance the national language policy of education, which states that the mother tongue or the language of the immediate community must be adopted as the language of education in primary schools, and English should only be introduced at a later stage. Other language policies in other domains that encourage the use of mother tongues are also not adopted so that the regional or national status of Yorùbá and other regional languages is theoretically but not fully implemented in practice [50].

Colonialism imported the English language to Nigeria and English has since been adopted as the official language in government, education and all official businesses. English is the language of the elite and fluency in English is synonymous with a good education. As a result, many parents, even those who are barely educated or not educated at all, ensure that their children are taught in English right from the elementary classes. In most schools, indigenous languages are referred to as vernacular and are prohibited. Violation usually attract fines and many times corporal punishment. Bilingualism is also believed to affect children's ability to attain competence in English and thus parents avoid speaking mother tongues at home for fear of raising children with poor communication in English. Many children therefore can neither speak, read nor write in Yorùbá and many do not even understand the language at all.

The great linguistic diversity in Nigeria also prevents people from speaking Yorùbá as often as they may have. Well over 450 languages are said to be spoken in Nigeria, therefore necessitating the use of English as the lingua franca. Less educated people use the Nigerian Pidgin, an English based pidgin, as a medium of communication, and some educated people use the Nigerian pidgin in less formal environments. [8].

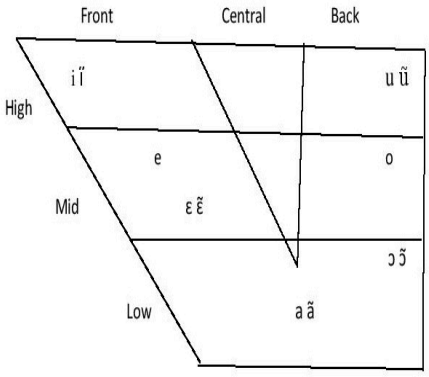
English is also the language spoken at offices, corporate organizations and the use of Yorùbá has greatly declined, resulting in lower competency among many users, code-mixing, code switching and several other language loss trends. In addition, although Yorùbá is still widely used in media and there exist radio and television programs in Yorùbá as well as newspapers, they receive their patronage from the older generation since there always exist the English versions which receive a greater patronage.

All the afore mentioned language situation have influenced the lack of adequate language development and consequently resulted in, resource scarcity of Yorùbá.

3.2 Phonology

The phonology of Yorùbá, which is how phonemes (unit of distinctive sound that account for meaning distinction) are strung together to form words, comprises of seven oral vowels, five nasal vowels, three syllabic nasals, seventeen consonants and two glides. Oral vowels are produced from the oral orifice while nasal vowels are produced through the nasal orifice. Nasalized vowels, those vowels produced with the nasal and oral orificies, are known to occur through an assimilation process when vowels precede nasal consonants. All the seven oral vowels can be nasalized.

Pitch, the perceptual correlate of vocal cords' frequency, is distinctive in Yorùbá. This distinctive or phonemic pitch is called tone. For example, the word rá(H) *disappear*, rà(L) *buy* and ra(M) *rub* are distinguished by the three tones in Yoruba: High-tone, mid-tone and Low-tone. High-tone is orthographically represented with acute accent, the Low-tone with grave accent and the mid is unmarked. These tones can interact to form rising and falling tones for instance yìí *this* and náà *the*. Vowels and syllabic nasals are the tone bearing units in Yorùbá. We have been able to use tones in addition to other features in our model to induce conditional properties. Details can be found in 6.3.7. Consonant clusters are not permitted in Yorùbá but long vowels are possible.



		Labial	Alveolar	Palatal	Velar	Labiovelar	Glottal
stops	Voiceless		t		k	k͡p	
	Voiced	b	d		g	g͡b	
Affricates	Voiced			ʤ			
Fricatives	Voiceless	f	s	ʃ			
Nasals		m	n	ŋ			
Approximants	lateral		l				
	Central		r	j		w	h

Figure 3.1: Yorùbá Vowel and Consonant Chart.

In Figure 3.1, the vowel chart is made of both oral and nasalized vowels. The consonant chart which occurs below the vowel chart contains the phonemes. Allophones are not represented.

3.3 Morphology

Typologically, Yorùbá is an analytic language, that is, grammatical relations are expressed with little or no use of inflection. Tense is not marked morphologically and agreement relation is limited. It is a combination of *isolating*, that is, it has a low morpheme per word ratio, and *agglutinating* through the use of prefixes, interfixes, and reduplication. Prefixation and reduplication are the most common word formation processes in Yorùbá. Word formation processes in Yorùbá are also derivational and words changed through inflections are usually through tonal variations.

Prefixation is very dominant in the word formation process of Yorùbá. There is a set of prefixes for verbs and nouns and there are prefixes for converting verbs to nominals through a process called deverbalization. For instance, there is a prefix *oní* which works with nouns to indicate possession, property or making of the particular noun. This prefix is a combination of two prefixes *o* 'he/she/it' and *ní* 'to have'. For example when the prefix

oní 'to have' is attached to a noun like *bàtà* 'shoe', as in *oníbàtà*, it can mean 'shoe seller, shoe maker or shoe owner'.

Compounding is another dominant word formation process. Due to the syllable structure, compounding usually involves vowel ellision, coalescense, epenthesis, and compensatory lengthening to resolve vowel hiatus and consonant clusters. During compounding: (a) vowel assimilation may or may not occur. For example, the two words *ọmọ* 'child' and *àlè* 'concubine' can compound to give either *ọmọ-àlè* 'illegitimate child' (without assimilation), or *ọmaàlè* 'incorrigible person' (with assimilation). (b) one of the two vowels at the morpheme boundary may or may not elide. For example the two words *ọmọ* 'child' and ilé 'house' can compound to give either *ọmọ ilé* 'child in the house' (without elision), or *ọmọ* 'child' lé 'wall gecko or small lizard' with vowel elision leading to contraction [18].

Reduplication is a prominent word formation process in Yorùbá. There are two types of reduplication in the Yorùbá language – full or partial reduplication [18]. Full reduplication of nouns can give the meanings of extentiveness or degree. For example *ọmọ-ọmọ* 'grandchild, descendant'; *ilé-ilé* 'every house/home'; etc. It can also be used for plural marking in the reduplication of a modifier. For instance *ilé ńlá* 'big house' ilé nla *ńlá* 'big houses' (more on this in section 3.5). Partial reduplication occurs where a part of the stem is copied either before the stem or after the stem. Examples are *pa* 'kill', *pípa* 'to kill'; *rá* 'buy', *rírá* 'buying'; where the copying is to the front of the stem.

Interfixation is another word formation process in Yorùbá, where an affix is added between two word. Examples are *ọmọkọ́mọ* 'child any child', *ilékílé* 'house any house'.

3.4 Orthography

Orthography refers to the standardized writing system of a language and it consists of the symbols and conventions used in writing a language. An orthography should be consistent, familiar to the users with the symbols adopted, it must have different ways to write all different significant sounds of the language and it should be convenient to use.

A standard orthography is therefore, a convention which is learned, consistent, stable and accepted as correct across a community. It is a set of conventions which are observed by all writers and it is used in formal education. The Yorùbá language has been written since the early 19th century. However, there have been several changes in the orthography since then. [52, 72].

Orthographically, the Yorùbá alphabet consists of 25 letters and their variant forms i.e. capital, small letter, spelling system and a set of punctuation marks. It also uses the familiar Latin characters. Nasalized vowels are represented with a 'n' after the vowel. Syllabic nasals are written as 'n' except when they occur before 'b' where they are represented as 'm'. Although tones occur on syllables, there are orthographically marked on vowels and

syllabic nasals. Yorùbá syllables are open; this means they end in vowels. The syllables are formed by a single vowel, by consonant or glide plus vowel or a syllabic nasal.

The present standards were established in 1974, however, there remains a great deal of contention over writing conventions, spelling, grammar, the use of tone marks and some other linguistic formalities. For the purpose of this research, we have adopted the generally accepted formalities of the standard language according to grammars by Bamgbose and Awobuluyi [17, 20]. This includes full tone marking including tone marks on syllabic nasals.

Figure 3.2 summarizes the letters used in Yorùbá orthography and the sounds they represent. The letters are written in bold and the sounds that they represent are directly below them in square brackets.

Aa	**Bb**	**Dd**	**Ee**	**Ẹẹ**	**Ff**	**Gg**	**Gbgb**
[a]	[b]	[d]	[e]	[ɛ]	[f]	[g]	[gb]
Hh	**Ii**	**Jj**	**Kk**	**Ll**	**Mm**	**Nn**	**Oo**
[h]	[i]	[ʤ]	[k]	[l]	[m]	[n] [ŋ]	[o]
Ọọ	**Pp**	**Rr**	**Ss**	**Ṣṣ**	**Tt**	**Uu**	**Ww**
[ɔ]	[kp]	[r]	[s]	[ʃ]	[t]	[u]	[w]
Yy							
[j]							

Figure 3.2: The Yorùbá alphabet and the sounds they represent

3.5 Constituent Structure of Noun Phrases

As in English, the canonical word order is SVO as in ọmọ ra oúnjẹ (child buy food), but the word order is not as rigid as in English. However, there are are several interesting differences between the constituent structure of Yorùbá and English. Five of these are within the scope of noun phrases and have been handled in this work. Details of how we address these differences are in chapter 5.

1 In the arrangement of constituents, Yorùbá has a determiner final structure and determiners are not obligatory in noun phrases [10]. So that *ajá kékeré* 'dog small' and *ajá kékeré kan* 'dog small a' are felicitous in Yorùbá.

2 Yorùbá has a superstructure from which 'adjectives' are derived [17, 20, 27, 19, 73]. For instance the word *pupa* 'red' has two entries in the dictionary [19]. The first entry as a noun in *Pupa dára ju dúdú lọ* (red good more black than which means red is better than black) and as a verb in *ọmọ́ pupa* 'child red' which means 'child is red'. It is important to understand that in the example derived for the first entry, *pupa* and *dúdú* are nouns because both words can be replaced with the pronoun *'ó* 'he/she/it',

so that we can say *ó dára ju ú lọ*. The *ó* is derived as *ú* after *ju* through an assimilation process because Yorùbá abhors vowel hiatus. In situations of plural and honorific (just like *tu* and *vous* in french) nouns, we will have *Pupa dára ju àwọn dúdú lọ* 'red, good more plural, black than' *Pupa dára ju wọ́n lọ* and 'red good more them (or singular honorific) than)'. In the second example however, *ọmọ́ pupa* has also gone through a vowel deletion and elongation through an assimilation process which turned *ọmọ ó pupa* to *ọmọ́ pupa*. The mid tone *o* was deleted and the high tone *o* was elongated. Although the *ó* (which is a pronoun but functions as a determiner) in that sentence is implied but will not articulated by native speakers of Yorùbá. *ọmọ pupa* 'red child' however is an example where *pupa* performs a modifier or adjective function to the noun *ọmọ*. Several other words like kékeré 'small', *ńlá* 'big', *púpọ̀* 'plenty' are examples of some nouns or verbs that can function as adjectives. For the purpose of this paper, nouns or verbs that modify other nouns are referred to as adjectives and are used in their adjective function. Current model handles assimilation by representing the word twice. The first with assimilation and the other without assimilation. Future work will consider handling phonological processes such as assimilation in a more concise way.

3 In Yorùbá, agreement is morphologically absent and as a result, plurality is shown syntactically through the occurrence of a pronoun *àwọn* 'they' before a noun as in *àwọn ọmọ* 'they child' which means 'children'; addition of cardinals or adjectives after a noun as in *ọmọ méjì* 'child two' implies 'two children', addition of an elided form of the pronoun *àwọn*, before a demonstrative as in *ọmọ wọ̀n yẹn* 'child they that means those children', the use of quantifiers which are inherently plural or reduplication of certain 'adjectives' after the noun as in *ajá dúdú dúdú* 'dog black black' [10]. It is important to note that the words used for plurality in Yorùbá have their own distinct meanings and are used in certain contexts to depict plurality. For instance *Ilé ńlá ńlá* 'house big big' will denote several houses but *esè kòǹgbà kòǹgbà* 'leg big big', which means a very big leg, will not be a plural expression despite both phrases having identical categories [73]. This shows how reduplication does not always result in plurality.

4 Yorùbá does not have a formal feature for indicating gender and when gender distinction is mentioned in Yorùbá grammar, it is a translation equivalence or it is with reference to the structure of English and not Yorùbá [20]. The use of words like *akọ* 'male', *abo* 'female', *obìnrin* 'female human being', *ọkùnrin* 'male human being' can be used to express gender in words like: *Abo kìnìhún* 'lioness', *ọmọ obìnrì* 'child female or daughter', *akọ ẹlẹ́dẹ̀* 'pig' while the pronoun "o" is used to refer to 'he/she/it' [20]. We explain how we use gender features in section 5.2.7 and 5.3.7.

5 Another interesting distinction between Yorùbá and English structure is the position of adjectives in phrases. Many adjectives occur after nouns, but it is also common to

have adjectives before the noun. For instance, *òkú* 'dead' always occurs before a noun as in *òkú ẹran* 'dead meat'. Adjectives that describe a part of something always occur before a noun as in *ìdajì oúnjẹ* 'half food'. Adjectives that describe the attribute of a person or thing usually occurs before a noun as in *kékeré ẹkùn* 'small leopard', but there are some cases where having the adjective after the noun are accepted as correct [73], as in the example in item 3. We are able to handle these different arrangements with our model's ability to induce conditional properties described in 5.3.7 and our results (in chapter 7) are consistent with this linguistic claims.

Although, we cannot discount language universalities, these peculiarities which can cause difficulty in simple comparisons have been managed in very interesting ways.

Chapter 4

Linguistic Details

4.1 Data Collection

Data collection for this research has been a combination of introspective and empirical collection methods. A representative subset of Yorùbá noun phrases were developed by native speakers of Yorùbá, some of who are linguists, and also, by the author's introspection. These phrases were analysed by the author together with other linguists to determine their part of speech tags (details in 3.5 and 4.2). In all, fourty five phrases (details in Appendix D) were generated to be used as input phrases in the model to include at least one example of all the different grammatical structures possible in the subset of Yorùbá noun phrases we considered. A property grammar of English which covers the subset of the Yorùbá noun phrases was created by the author using some books and scientific papers[1, 61, 62, 80] and we further cross-checked with linguists.

A context-free grammar was also developed for evaluation purposes; to enable us to compare the output of the model with the context-free grammar. This grammar was developed by linguists who have native competence in Yorùbá. The first context-free grammar was developed by the author who is also a linguist and the grammar was extended (to include other structures that were omitted) by the other linguists whose names are Demola Lewis, Tunde Adegbola and Samuel Akinbo. The grammar was then used to generate noun phrases. Native speakers of Yorùbá who do not have formal linguistic training were invited to check these phrases generated by the context-free grammar in order to ascertain that those phrases are intuitively correct to the regular user of the language.

We developed a context-free grammar because there is no known treebank of Yorùbá and Penn Treebank only has a dictionary of words. This approach of data collection was employed in order to ensure that our model is realistic, correct and robust. Introspection has proven to be the most reliable process of data collection and also very useful for building models which require high level linguistic competence such as this WG model [34]. Introspection as previously mentioned, has been that of the author who is a native speaker of

Yorùbá and also has formal training in linguistics. As we aforementioned, we have double-checked our introspective analysis of the data by consulting as well seven other native speakers of Yorùbá, three of which also have formal graduate level training in linguistics.

Analysed data have also been compared with two exisiting grammar description of Yorùbá. The first by Ayò Bámgbósé [20] and the other by Awóbùlúyí [17]. It was important to compare our analysis with these exisiting grammatical descriptions of Yorùbá, considering that they are one of the earliest contributions of native speakers who have formal linguistic training to the description of the Yorùbá grammar [83]. Although, there were several similarities between our data analysis and those in the grammar, we were able to identify differences too. The differences are majorly related to parts of speech tags assigned to words and the syntactic theory adopted for their analysis, abscence of gender and number in analysis, etc. For instance, [20] describes the head of the noun phrase as the first word occuring in a phrase, so that in a phrase like: *àwọn ọm ọ* (they child) which means children, *àwọn* will be the head of the phrase and *ọm ọ*(child) will be a qualifier, whereas, we analyze *àwọn*(they) as a plural marker and *ọm ọ*(child) as the head. However, [73]'s description has been useful to back up our analysis, especially with regards to conditional precedence properties induced, while, [10]'s description of Yorùbá noun phrases has also been very useful especially in relation to plural marking.

4.2 Strategies for Part of Speech Parsing

Accuracy of part-of-speech tagging is a critical and fundamental building block for many computational linguistic tasks including grammar induction tasks. Assigning correct part-of-speech tags to each input word explicitly indicates some inherent grammatical structure of any language and a wrong part-of-speech tag will distort the grammatical structure of a language. Rule-based (derived from linguistic rules) [28], data-driven (derived from statistical analysis of language data) [66] and hybrid methods (which are a combination of rule driven and data driven approaches) [87] have been used extensively in literature for part-of-speech tagging [95]. Data-driven approaches have yielded very good results, albeit for Indo-European languages and other data rich languages like English, Dutch, German, etc. However, data driven approaches have error rates which are usually reducible by only a few percentage and also work poorly with languages which have a less fixed word order [91]. Rule-based part-of-speech tagging on the other hand is the oldest approach that uses hand-written rules for tagging. Rule based taggers depend on dictionaries or lexica that contain word forms together with their associated part-of-speech labels as well as context sensitive rules to choose the appropriate tags during application. The dictionary or lexicon is consulted to get possible tags for each word. Hand-written context sensitive rules are used to identify the correct tag if a word has more than one possible tag. The linguistic features of the word and other words surrounding it are analyzed for disambiguation purposes. For

example, a tagger for English will have a rule such as: if the a word in a phrase is a determiner then a noun must be present somewhere in the phrase.

We adopt a rule-based approach. The tagsets were developed with the use of the [19] of Yorùbá as well as native speakers of Yorùbá who have formal linguistic training. The tagsets were compared to the grammars of Ayò Bámgbósé [20] and Awóbùlúyí [17], one of the earliest grammar descriptions of Yorùbá by native speakers who have formal linguistic training. We also used [10] whose grammatical description of Yorùbá is more recent.

We use the following tagsets:

i noun - ajá (dog), ẹran (goat)

ii pronoun - àwọn (they), ìwọ (you)

iii proper-noun - Ayò, Bámgbósé

iv determiner - kan (a), náà(the)

v quantifier - gbogbo(every), ìdajì(half)

vi adjective - dúdú(black), pupa(red)

We further define features for each word in order to provide a fine-grained definition to each word tag. We use the following features:

i Number - We define number as either singular, plural or dual. Dual number refers to those words that can be both singular and plural.

ii Gender - We define feminine, masculine and neuter gender.

iii Tone - The mid, high and low tones are used as features. The tones of a word are stored in a list which contains the tones on each syllable in that word.

iv Person - The first, second, third persons are features we use to define pronouns.

v Definitiveness- Determiners are defined as either indefinite or definite. This is in addition to other features relevant to determiners.

vi Type - Nouns are defined as common, collective, abstract, and concrete. Adjectives are divided into attributive, descriptive, quantity and number. Quantity is further divided into countable and uncountable and descriptive is further divided into colour and size. All categories are also classified as animate, inanimate or both. Those which are labelled as both, refer to categories that can be used for both animate and inanimate objects.

These features have been carefully chosen to ensure that our model accounts for the unique traits of Yorùbá.

4.3 Property Grammars

The idea of constraint is present in modern linguistic theories such as Lexical Functional grammars (LFG) and Head-driven Phrase Structure grammars (HPSG) [76]. However, constraint satisfaction, a way of implementing constraints, is not really incoporated and is not the basis of the implementation of these theories. We use a formalism called Property Grammar (PG)[22] which is based completely on constraints: all linguistic information is represented as properties between pairs of constituents, which allow parsing to be implemented as a constraint satisfaction problem. The set of properties forms a system of constraints expressed over categories. They represent different kinds of information (e.g. agreement, morphology or semantics) that can be used typically for contextual restrictions. PG differs from HPSG and LFG in that it does not belong to the generative syntax family. HPSG belongs somewhere between the proof-theoretic and model-theoretic syntax and LFG to the model-theoretic syntax one [24].

Constraints represent information and the first benefit of representing a problem using constraints is that of partially solving it, so this allows for partial solutions to be found even when a full solution is not available. In addition, constraints can be specified at any level of the linguistic description: they can represent local properties, such as those for lexical selection, or universal ones.

These *constraint-based* or *property-based* theories, such as Property Grammars (PG) [23] evolved from Immediate Dominance/Linear Precedence (IDLP), which unfolds a rewrite rule into the two constraints: (1) of immediate dominance (expressing which categories are allowable daughters of a phrasal category) and (2) linear precedence (expressing which of the daughters must precede which others).

For example in the PG framework, English noun phrases can be described using constraints such as: precedence (a determiner must precede a noun, an adjective must precede a noun), requirement (a singular non-generic noun must be used with a determiner), obligation (a noun phrase must contain the head noun), and so on. The linguistic structure (e.g phrase, sentence, clause, etc) is characterized by a list of the constraints it satisfies and a list of constraints it violates, so that even incorrect or incomplete phrases will be parsed. Using PGs also enables us to relax certain constraints by declaring conditions under which those constraints should be relaxed. For instance, the obligation rule of English can be relaxed to allow pronouns and proper nouns instead of only a noun because pronouns and proper nouns can also function as the head of a phrase. [44] encodes the input PG into a set of Constraint Handling Rule Grammars (CHRG) rules that directly interpret the grammar in terms of satisfied or relaxed constraints and a syntactic tree is the implicit result. Womb Grammars which is an adaptation of this framework into grammar transformation [42], [35] induces a language's syntactic structure from that of another.

In theories that use constraint satisfaction, the modularity obtained by modelling grammatical information as constraints leads to more *robust parsers*, since it allows us to explicitly identify from the parser's output which constraints are satisfied and which fail. As an effect, even incomplete or incorrect sentences can be accepted, instead of silently failing to be parsed. We can also produce some indication of the sentence's degree of acceptability by analyzing the failed properties.

The PG formalism presently comprises the following seven categories (we adopt the handy notation of [47], and the same example):

Constituency $A : S$, children must have categories in the set S

Obligation $A : \triangle B$, at least one B child

Uniqueness $A : B!$, at most one B child

Precedence $A : B \prec C$, B children precede C children

Requirement $A : B \Rightarrow C$, if B is a child, then also C is a child

Exclusion $A : B \not\Rightarrow C$, B and C children are mutually exclusive

Dependency $A : B \sim C$, the features of B and C are the same

Example 1. *For example, if we denote determiners by D, nouns by N, personal nouns by PN, verbs by V, noun phrases by NP, and verb phrases by VP , the context-free rules* $NP \rightarrow D\ N$ *and* $NP \rightarrow N$*, which determine what a noun phrase is, can be translated into the following equivalent constraints:* $NP : \{D, N\}$*,* $NP : D!$*,* $NP : \triangle N$*,* $NP : N!$*,* $NP : D \prec N$*,* $D : \{\}$*,* $N : \{\}$*.*

In the PG formalism, A refers to the phrase or sentence being defined. In our case, A refers to Noun phrases, B and C are categories such as nouns, adjectives, etc. These properties contain the basic syntactic information. The use of a dependency property may make it possible to express certain features of a dependency grammar using the formalism of property grammars. This is possible because, just like dependency grammars, dependency property is concerned with syntactico-semantic relations. For instance np(dependence(adjective,noun)) is concerned with using features (described in 4.2) to determine the gender and number relationship between adjectives and nouns and other dependency information can be modelled in similar fashion. The features are introduced for each word in the lexicon (details in section 5.2) and all the subsets of categories involved in the description of a given input can be characterized by the constraint system (i.e. the grammar). The constituency property contains dependency information that tell us each category can be associated to another.

Thus, there is a general notion of characterization formed by the set of properties that succeed and those that fail which together characterize an input. The characteristics of an input corresponds to the result derived from the constraint system at the end of the parse. The result of a parsed property is either satisfied or unsatisfied and this allows (ungrammatical) inputs to still be parsed rather than just fail. It is defined as the set of properties that are satisfied plus the set of unsatisfied ones. The interest of such a conception is that every property in the grammar states some constraint on well formedness. It can be the case that all constraints are satisfied, but this is not an imperative condition. All kinds of input can consequently receive a characterization.

Chapter 5

The Womb Grammar Model of Grammar Induction

Womb Grammars was motivated by the need to describe the syntax of many minority languages which are not being studied because of a lack of adequate resources; these resources mainly being: linguistic data and linguists to analyse the data. The syntax of a *target* language is induced from the grammar of the *source* language and the *source* and *target* languages need not be closely related languages. The *source* language is the language which functions as the initiator of inference for learning the *target* language, which is, the language to be learned. A language's phrases are described in terms of constraints or properties between pairs of direct daughters of a phrasal category referred to as properties. The parsing capabilities inherent in these properties are extended into a grammatical induction and parsing model [42].

There are two versions of WGs: *Hybrid Womb Grammars*, in which the source language is an existing human language for which the syntax is known, and Universal Womb Grammars, in which the source syntax is a hypothetical universal grammar created by the author, and it contains all possible properties between pairs of constituents [42]. The *Hybrid Womb Grammars* can use any language whose syntax has been studied by linguists as its source language. We use a hybrid model in this work and English as the source language.. A *Universal Womb Grammars* version on the other hand use a source grammar for a nonexistent language which can be created to include all plausible properties which encompass all human languages. For instance, while an English source grammar will have rules that *a determiner precedes a noun*, a universal grammar will have rules that *a determiner precedes a noun* and the converse, *a noun precedes a determiner*. Parsing with WGs has the novel approach of using constraint solving to address a problem which is more usually solved as a machine learning problem.

In this work, the grammar of our source language, English, is described as properties between constituents as described by [23]. We use the hybrid version of Womb Grammars

and not the universal version, to which we feed the following English grammar (the code notation can be found in appendix c):

Constituency NP : *determiner, noun, adjective, pronoun, proper-noun, quantifier*;

Obligation $NP : \triangle$*noun,pronoun,proper noun*

Precedence NP : *determiner* $\prec$ *noun*; NP : *determiner* $\prec$ *adjective*; NP : *pronoun* $\prec$ *noun*; NP : *pronoun* $\prec$ *adjective*; NP : *adjective* $\prec$ *noun*; NP : *quantifier* $\prec$ *determiner*; NP : *quantifier* $\prec$ *pronoun*; NP : *quantifier* $\prec$ *adjective*; NP : *quantifier* $\prec$ *noun*

Requirement NP : *noun* $\Rightarrow$ *determiner*

Dependency NP : *quantifier* $\sim$ *noun*; NP : *adjective* $\sim$ *noun*; NP : *determiner* $\sim$ *noun*

WG extends the parsing capabilities implicit in these properties into a model of grammatical induction, in addition to parsing. The previous implementation of WGs [42] allowed greater efficiency by adopting a failure-driven approach where only failed constraints were analyzed, rather than explicitly calculating all successful and unsuccessful constraints. However in adapting the model into Yorùbá, we have found it more efficient to explicitly calculate satisfied constraints as well. This is partly because in order to arrive at as nuanced a description as needed for Yorùbá, we had to extend the Property Grammar model to accommodate what we call conditional constraints: those constraints which have failed in some phrases and succeeded in others, but also have a unique pattern responsible for this behaviour (more on this later).

The general WG model can be described as follows: Let L^S be the source language and its syntactic structure be referred to as L^S_{syntax}. We will call the target language L^T and its lexicon will be (L^T_{lex}). A correct and sufficiently representative set of phrases in L^T can be fed into a hybrid parser consisting of L^S_{syntax} and L^T_{lex}. A correct and sufficiently representative set of phrases will consist of at least one example of all plausible syntactic structures. Parsing the sentences may result in some of the sentences being marked as unsatisfied by the parser and an analysis of the constraints which these "unsatisfied" sentences violate can subsequently disclose how to repair the L^S_{syntax} so that it accepts as satisfied, the sentences in the corpus of L_T which were previously unsatisfied. This process transforms L^S_{syntax} into L^T_{syntax} by modifying the constraints that were violated into constraints that accept the input. Figures 6.1 and 6.2 respectively show the problem and solution in schematic form adapted from [42].

An Example

Let $L^S = English$ and $L^T =$ Yorùbá, and for example, let English determiners always precede the nouns they modify, while in Yorùbá they always follow it (this could be an

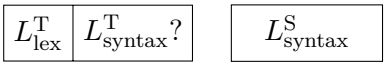


Figure 5.1: The Problem

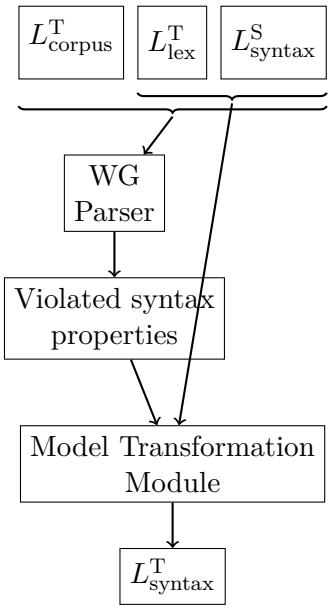


Figure 5.2: The Solution

oversimplification, just for illustration purposes). Thus "a black book" is correct English, whereas in Yorùbá we would more readily say "ìwé dúdú kan" (book, black, a).

If we insert the Yorùbá lexicon and the English syntax properties into our WG parser, and run a representative input of correct Yorùbá noun phrases by the hybrid WG parser, the precedence property will be declared unsatisfied when parsing phrases such as "ìwé dúdú kan". The model transformation module can then look at the list of unsatisfied properties, and produce the previously unknown syntactic structure of L^T's parser by modifying the properties in L^S_{syntax} so that there are no more violations in the input phrases of the target language.

Some of the required modifications may be easy to identify and implement. For instance, to accept "ìwé dúdú kan" the English rule that determiners precede nouns (noted $det < n$) needs to be deleted. However, subtler modifications may be required, after some statistical analysis: if in our L^T corpus, which we assume to be representative, *all* determiners appear after the noun they modify, Yorùbá will in addition to deleting the English precedence property: $det < n$, add the reverse precedence property of English: $n < det$.

5.0.1 Constraint Handling Rule Grammars (CHRG)

Constraint Satisfaction has produced impressive results in many Artificial intelligence areas, including human language processing. To a large extent, this powerful problem-solving paradigm serves as a bridge between languages for problem description and those for problem solving, a disjoin that has been prevalent in the field until recently.

Constraint Handling Rule Grammars (CHRG) are a constraint-based grammar formalism added on top of Constraint Handling Rules (CHR) [88] similar to the way Definite Clause Grammars are implemented over Prolog. A CHRG works as a bottom-up parser, and the formalism provides several advantages for natural language analysis. A notable advantage is that the notation supports context-sensitive rules that may consider arbitrary symbols to the left and right of a sequence [35].

CHRG (CHR Grammars) strives to incorporate as much as possible of CHR's expressibility. This includes propagation rules, simplification and simpagation rules and the capacity for a grammar rule to make considerations for arbitrary grammar symbols that may occur to the left and right of a sequence that is supposed to match a given nonterminal. Ambiguity is not a problem for CHRG, all different parses are evaluated in parallel in the same constraint store - without backtracking. This results in tagger-like grammar rules that can be used for disambiguating simple and ambiguous context-free grammar rules, and provides also a way to handle coordination in natural language. In case of errors, the parser is robust enough to deliver as its result those subphrases that have been recognized. Since the advent of CHR [88] and of its grammatical counterpart CHRG [35], constraint-based linguistic formalisms can materialize through fairly direct methodologies.

In a nutshell, CHRG rules rewrite constraints into other constraints, that are subject to possible checks described in a rule's guard and stated as Prolog calls. Their general format is:

$$\alpha \texttt{ -\ } \beta \texttt{ /- } \gamma \texttt{ ::> } G \texttt{ | } \delta \texttt{.}$$

This rule is called a propagation rule and it means, if α occurs before β, and β before γ, and G condition is fulfilled, then store δ in the constraint store. The part of the rule preceding the arrow `::>` is called the head, G the guard, and δ the body; $\alpha, \beta, \gamma, \delta$ are sequences of grammar symbols and constraints so that β contains at least one grammar symbol, and δ contains exactly one grammar symbol which is a nonterminal (and perhaps constraints); α (γ) is called *left (right) context* and β the *core* of the head; G is a conjunction of built-in constraints as in CHR and no variable in G can occur in δ. If left or right context is empty, the corresponding marker is left out and if G is empty (interpreted as **`true`**), the vertical bar is left out. The convention from DCG is adopted that constraints (i.e., non-grammatical stuff) in head and body of a rule are enclosed by curly brackets. Special grammatical operators are provided for gaps and parallel matching (parallel matching is used when it is required that more than one pattern matches the string to be recognized). Gaps and parallel match are not allowed in rule bodies. A gap in the rule heads is noted "..." while parallel match is denoted by "$$". Gaps are used to establish references between two long distant elements [42].

A *simplification (grammar) rule* is similar to a propagation rule except that the arrow is replaced by `<:>` and the grammar symbols in the head are deleted. A *simpagation*

(grammar) rule is similar to a simplification rule, except that one or more grammar symbols in its head are prefixed by a ! symbol, which means that such a grammar symbol is not removed. It can also be fully integrated with Prolog and a single file can conveniently incoporate prolog, CHR and CHRG codes.

An example with propagation, simplification and simpagation rules follows:

```
:- compile(chrg).
:- chrg_symbols np/0, adjective/0, verb/0, sentence/0.

np, verb, np ::> sentence.
adjective, np, verb, np <:> sentence.
!np, verb, adjective, np <:> sentence.
[dogs] ::> np.
[bones] ::> np.
[like] ::> verb.
[big] ::> adjective.

end_of_CHRG_source.
```

Figure 5.3: A CHRG example, showing propagation, simplification and simpagation rules

In Figure 5.3, np/0, verb/0, adjective/0 and sentence/0 declare grammar symbols of arity 0 and so it can be used in the rules that follow. The first rule says that if a sequence of np, verb and np occurs, then, a representation of sentence is added to the store. The second rule states that if a sequence of adjective, np, verb and np occurs, then, a sentence is added to the store and all symbols to the left of the sentence are deleted. The third rule is like the second, however, the initial np is not deleted.

In order to execute the parser, we need only to call the parse predicate, a built-in prolog predicate, with a list of Prolog atoms in the following way:

```
?- parse([big, dogs, like, bones]).
<0> big <1> dogs <2> like <3> bones <4>
token(3,4,bones)
token(2,3,like)
token(1,2,dogs)
token(0,1,big)
all(0,4)
sentence(0,4)
sentence(1,4)
```

Figure 5.4: Executing the parser

Our results in Figure 5.4, show the word boundaries, phrase boundaries, and also tell us that two sentences were discovered: the first one being the entire sentence *big dogs like bones*, and the second one *dogs like bones*.

Constraint solving as an implementation paradigm for WGs allows us to convey linguistic constraints and also test them in a modular fashion. The results are also concise.

5.1 Some Implementation Details

Our CHRG implementation adapts the original implementation [35] where the appropriate WG constraints were entered in terms of a constraint g/1, whose argument stored each possible grammar property. For instance, our English grammar hybrid parser for noun phrases includes the constraints below (some of these constraints are also in [42], although we wrap each property in np) and some others in appendix C:

g(np(obligatority([noun, proper-noun, pronoun]))) - at least one *noun or pronoun or proper-noun* child

g(np(constituency(determiner))) - *determiner* children must have categories in the set

g(np(constituency(adjective))) - *adjective* children must have categories in the set

g(np(precedence(determiner,adjective))) - *determiner* children precede *adjective* children

g(np(precedence(determiner,noun))) - *determiner* children precede *noun* children

g(np(requirement(noun, determiner))) - if *noun* is a child, then also *determiner* is a child

g(np(dependence(determiner,noun))) - the features of determiner and noun are the same

and so on.

In [42], these properties are weeded out upon detection of a violation by CHRG rules that look for them e.g. an input noun phrase where a noun precedes an adjective may provoke deletion of the constraint g(precedence(adjective, noun)) when the following CHRG rule applies:

```
!word(C2,_,_), ... , !word(C1,_,_),
{g(precedence(C1,C2))} <:>
{update(precedence(C1,C2))}.
```

Figure 5.5: A Simpagation rule from the previous implementation

The parser worked by storing each word in the lexicon in a CHRG symbol word/3 along with its category and traits which we now refer to as features, as in:

```
word(determiner,[singular,masculine],le).
```

Only gender and number features were used in previous versions. Since the CHRG parse predicate stores the position of each word in the sentence, this simpagation rule in *Figure 6.3* is triggered when a word of category *C2* comes before category *C1*, given the existence of the grammar constraint that *C1* must precede *C2*. The precedence property will fail in this case and *update* stores this property accordingly [1].

When the parse predicate is invoked, the constraints of the source grammar are tested with the input phrases of the target language. The properties are tested on input phrases only once, so that if *15* input phrases are to be tested and all properties have been found and tested in the first *5* phrases, the remaining *10* phrases do not need to be tested for any properties. This tests are done bottom-up. Constraints that are violated are output as unsatisfied. Other constraints, those which did not fail and those irrelevant to the target language were printed.

5.2 Methodology

In our model, we store each Yorùbá word in the lexicon in a CHRG symbol word/3 along with its category and features (i.e. word(adjective,[dual,neuter,descriptive-colour,both,[mid-mid]],funfun).).

[1] "Recall that in CHRG syntax the symbols prefixed with exclamation points are kept, while the ones without are replaced by the body of the rule, in this case an update constraint that invokes some housekeeping procedures." [4]

```
!word(C2,F1,_):(N1,_),...,
!word(C1,F2,_):(_, N4),
{g(np(precedence(C1,C2)))}, {all(A, B)}<:>
{fail(np(precedence(C1,C2)), F1, F2, N1, N4, A, B)}.
```

Figure 5.6: A Simpagation rule in our model

This simpagation rule in *Figure 5.6* is triggered when a word of category C2 comes before category C1, given the existence of the grammar constraint that C1 must precede C2 and the phrase boundaries in which the word is found. The property will fail and the property *(np(precedence(C1,C2))*, features *F1,F2*, the word boundaries *N1, N4* and the phrase boundaries *A,B* will be stored.

For each phrase, constraints that are violated at least once are printed as unsatisfied with the relevant word and phrase boundaries. Constraints that are satisfied on the other hand are output as satisfied with the relevant word and phrase boundaries as well. We maintain two lists, one which has all properties that are unsatisfied at least once with the features and the number of times they fail as well as a second list which saves the properties that are satisfied at least once with the number of times they succeed. The information in these two lists are useful during the induction phase of our work. We provide more details of our methodology in the following subsections.

5.2.1 Modified parsing that calculates both failure and success explicitly

Previous models of WG focused on failure driven parsing, under the assumption that failed properties are usually the complement of those satisfied i.e., in order to improve efficiency they calculated only the properties that failed. This implies that they can be derived from the failed ones if needed and in general, a grammar is induced by repairing failures. However, our more in depth analysis in the context of Yorùbá has uncovered the need for more detail than simply failing or succeeding, as in the case of conditional properties. We therefore now use a success conscious *and* failure conscious approach for inducing the grammar of our target language, Yorùbá. Each input phrase of the target language is tested with all relevant constraints for both failure and success. This makes the model slightly less efficient than if we only were to calculate failed properties, but of course the gain is in accuracy.

Efficiency is still guaranteed by the normal CHRG way of operating: rules will only trigger when relevant, for example, if a phrase is comprised of only a noun and an adjective, it will not be tested with for instance precedence(pronoun, determiner) or any other constraint whose categories are different from those of the input phrase. In addition, if a property fails in a particular phrase, it will not be tested for success again and vice versa. This is because a property cannot succeed and fail at the same time.

We keep a list of all properties that fail and another for those that succeed, together with the features of the categories of each input phrase as well as their counts.

It is important to state that constituency constraints are tested only for success. This is because we are interested in checking that our target grammar shares similar constituents with our source language and testing for failure will be irrelevant for these constraints. We also are able to test for some properties that are present in the target grammar but are not in the source grammar. We also test for success alone in such cases. More of this in section 5.2.6.

5.2.2 How we save failed and succeeded properties

We store failed properties and succeeded properties separately and we save each property together with the categories, features, word boundaries and phrase boundaries. We first check if a property is already in the list so that we do not duplicate properties. If the property already occurs, we check if the features of the property to be added are different from the features already in the list. The new features of the property to be added are added to the property and the counter for the property and the features is incremented if the feature does not already occur in the list. If it is already in the list, we only increment the counter for the features and property respectively.

```
    incnum(Count, Ncount):- Ncount is Count + 1.


succeed(Prop,Features1, Features2, N1, N2, A, B),

succeededproperties(_,List), succeededboundary(_, List1)<=>

member([Prop, FeaturesPairList,CountP],List),

not(member([Features1, Features2,_], FeaturesPairList)),

succeededboundary(_, [[Prop, N1, N2, A, B]|List1]),

delete(List, [Prop, FeaturesPairList,CountP], Nlist),

incnum(CountP, NcountP)|

succeededproperties(_,[[Prop, [[Features1, Features2, 1]|

FeaturesPairList],NcountP]|Nlist]).

succeed(Prop, Features1, Features2, N1, N2, A, B),

succeededproperties(_Prop_List,List),

succeededboundary(_, List1) <=>

succeededproperties(_X,[[Prop,[[Features1, Features2, 1]], 1] | List]),

succeededboundary(_, [[Prop, N1, N2, A, B]|List1]).
```

Figure 5.7: Some rules to save succeeded propeties

In *figure 5.7*, if there is a *succeed/7*, *succeededproperties/2* and *succeededboundary/2*, we test if the features of the property are members of the *succeededproperties* list. If we find that the *features* are already in the list, we delete the property, features and the count; increment the count for that property and update the property the features and the count. If the list is empty, the property and features are added to the list and the count is equal to one.

Incnum is a helper predicate that we use to increment the count of properties or features.

5.2.3 How we handle tones

Yorùbá is a tone language, therefore it was imperative to ensure that tones are properly represented. We adopt a full tone marking approach (as described in section 3.2 and 3.4) so that each word is marked with its tones in order to avoid ambiguities. The tones are stored as features, such that for instance, the word tútù(cold) that has two syllables with a high and low tone on each syllable respectively is stored as "High-low" in the features of the word.

[tútù]::> $word(adjective, [dual, neuter, descriptive, both, [high - low]]$, **tútù).**

These tones are used to infer conditional properties in the second phase of parsing so that if a property is found to succeed only if it is around words with certain tones, such property will be said to be conditional. The conditions of such a property will be the tones which form a unique pattern in the success of the property. It is important to note that the tones are not used in isolation of other features and a property will be said to be conditioned on tones alone, if, tones are the only patterns responsible for its succeess or failure. The addition of tones makes our approach extensible to many other languages including tone languages.

5.2.4 Glossing of phrases

The system provides a record of the English gloss of every word in the target phrases and this record is consulted during parsing to produce the English equivalent for each word in the input phrases. The gloss is rendered below every phrase as in *figure 5.8*.

< 0 > àwọn < 1 > ọmọ < 2 > púpọ̀ < 3 >

```
they     child     plenty
```

Figure 5.8: Phrase transliterations

The gloss are provided to faciliate understanding especially of those who do not speak or understand Yorùbá. This will be useful for any other language which is not intelligible to the user, especially for linguists who often analyse languages that they can neither speak, read nor write.

5.2.5 Failed and satisfied properties

For each phrase parsed, the failed and or satisfied properties are explicitly output. This enables us to identify which specific properties are satisfied for each phrase and those which

are unsatisfied. We also explicitly retrieve the word boundaries of every word analysed during parsing as well as the phrase boundaries where the words are found. This is basically to eliminate ambiguity that can occur, in a situation where a single property fails, or succeeds more than once in a phrase; or if the model is extended to include other phrases for example a verb phrase. The boundaries before the semicolon represent the pair of words boundaries while the second boundary after the semi colon represents the phrasal bondaries.

In *Figure 5.9*, the parse results show that np(obligatority([noun,proper-noun,pronoun])) succeeds once because there is one noun in the phrase; np(constituency(determiner)) and np(constituency(noun)) succeed once each because the phrase has one noun and determiner; np(requirement(noun,determiner)) succeeds once because there is a noun and determiner in the phrase and, np(dependence(determiner,noun)) succeeds once, although the reason for the success of dependence is not visible here. np(precedence(determiner,noun)) fails once because the noun comes before the determiner which is different from the English ordering of determiner before nouns. We also printed boundaries beside each property with the word boundaries displayed immediately after the property and the phrasal boundaries just after semicolon.

< 0 > ilé-ìwé< 1 > náà < 2 >

```
school the

Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2

Succeeded Property: np(constituency(determiner))1-2; 0-2

Succeeded Property: np(constituency(noun))0-1; 0-2

Failed Property: np(precedence(determiner,noun))0-2; 0-2

Succeeded Property: np(requirement(noun,determiner))0-2; 0-2

Succeeded Property: np(dependence(determiner,noun))0-2; 0-2
```

Figure 5.9: Parse results of a sample phrase showing failed and succeeded properties

5.2.6 Inducing properties not present in the source language

WGs is also able to find all precedence and constituency constraints that are absent in our source but present in our target language.

```
prec(precedence(X,Y)):-
english_properties(List),
not(member(g(np(precedence(X,Y))), List)),
not(member(g(np(precedence(Y,X))), List)),
X \= Y.

word(C1,F1,_):(N1,_),...,word(C2,F2,_):(_,N4),
{others}, {all(A, B)}::> prec(precedence(C1,C2))
|{succeed(np(precedence(C1,C2)),F1, F2, N1, N4, A, B)}.
```

Figure 5.10: Inducing precedence properties present in target phrase but absent in source grammar

Figure 5.10 shows that during parsing, every pair of words with categories *C1* and *C2*, features *F1* and *F2* and word boundaries *N1* and *N2* are tested for precedence if and only if the precedence rule of those words' categories does not already exist in any precedence rule of the property grammar of English. We define a predicate *prec* which checks that the proposed precedence rule and its converse is not a member of the property grammar of English while we use *others*, a CHRG constraint to trigger this precedence rule. We do this in order to ensure that we do not create redundancies in a bid to infer precedence rules that occur only in the target language.

We are equally able to test the precedence constraints by checking if such a constraint succeeds in all relevant phrases and if the converse succeeds in any input phrase. If we find that the converse does not succeed in any of the input phrases, this constraint is induced as a property of the target language, else we search for a unique feature to ascertain if it should be induced as a conditional feature. In the case of constituency, like we explained in section 5.3.3, we are only interested in finding the constituent and the number of times it occurs although it is stored, is not relevant. If a constituent not present in the source grammar is found at least once, in the input phrase of the target language, then the constituent is induced as part of the grammar of the target language.

5.2.7 How we handle syntactic number and gender

As explained in section 3.5, number and gender are syntactically represented in Yorùbá. This feature is especially useful for the dependence property where we want to determine certain properties depend on another for gender and number. We also in section 4.2 explain that each word is defined with either singular, plural or dual number and feminine, masculine or neuter gender.

Before parsing, words which have a dual number and neuter gender accept the number and / or gender of the word that occurs before or after it if and only if those words do not

have dual number and / or neuter gender. After this update, the words are parsed with their updated features.

```
update(A,B,A,B).
word(_M,[N,G,_O,_P,_Q],_R)-\
word(noun,[dual,Y,Z,A,B],C),{g_update}<:>
N \= dual, G \= neuter, Y = neuter ->update(N, G, S, GR);
N \= dual, G = neuter, Y \= neuter->update(N,G,S,GR);
N \= dual, G = neuter, Y = neuter->update(N,G,S,GR)|
word(noun,[S,GR,Z,A,B],C).

word(noun,[dual,Y,O,P,Q],C)/-
word(_,[N,G,_,_,_],_),{g_update}<:>
N \= dual, G \= neuter, Y = neuter->update(N, G, S, GR);
N \= dual, G = neuter, Y \= neuter->update(N, G, S, GR);
N \= dual, G = neuter, Y = neuter->update(N, G, S, GR)|
word(noun,[S,GR,O,P,Q],C).
```

Figure 5.11: Rules for updating the number and / or gender features of a word

The first rule in *Figure 5.11* is used to update nouns which occur before any word which does not have dual number and / or neuter gender and if the constraint to update the gender and number is in the constraint store. In the first condition, if the number feature of the word preceding the noun is not dual and the gender feature is not neuter and if the gender feature of the noun is neuter, then the noun is updated with the number and gender features of the preceding word. We do this by assigning the number value X of the first word to S and the gender value N to the G which is now stored as the updated value of the noun. In the second condition and third conditions, if the word does not have a dual number but has neuter gender, the number feature of the noun is only updated but the gender remains the same. In this case, we update the value as in the first condition, except that we assign the gender value of the noun to G. The second rule is like the first, however, the noun occurs before the word that can be used to update its number and / or gender features.

5.2.8 Parsing each property in our English subset

Parsing with dependence constraints

Due to the peculiarities of Yorùbá grammar with regards to gender and number, detailed in section 3.5, we initialize all words with neuter gender and dual number, save a few words

which have their number inherent in their meaning and function (words such as *ọ̀pọ̀lọpọ̀* 'numerous', *àwọn* 'third person plural'. We also implement a rule (*Figure 5.11*) that creates a variant for each number or gender tag based on the environment in which the word is found. This successfully handles the syntactic process of defining number and gender in Yorùbá. The dependency rule of English is then tested for satisfaction and otherwise. This parsing strategy clearly accomodates the existence of plurality in Yorùbá without a previous knowledge of the peculiarities of Yorùbá's syntactic strategies for number, as well as gender which is in reference to English. However, though plurality and gender are present in Yorùbá grammar and are seen to succeed in parse results, the presence of failed dependence relations is further analyzed to determine if the dependency constraints should be a conditional property in Yorùbá. Results show that there is no pattern under which failure or success occurs. This lays credence to research findings by linguists who have studied the Yorùbá grammar [17, 20, 10].

```
word(X,[F1,A,B,C,D],_):(N1,_),...,word(Y,[F2,E,F,G,H],_):(_,N4),
{g(np(dependence(C1,C2)))}, {all(NA,NB)} ::>
X=C1, Y= C2, F1=F2-> assign(X, [F1,A,B,C,D], Y, [F2,E,F,G,H] ,
Rg, FRg, Rd, FRd);
X=C2, Y= C1, F1=F2 ->assign(Y,[F2,E,F,G,H],X,[F1,A,B,C,D],Rg,FRg,Rd,FRd) |
{succeed(np(dependence(Rg, Rd)), FRg, FRd, N1, N4,NA,NB)}.

word(X,[F1,A,B,C,D],_):(N1,_),...,word(Y,[F2,E,F,G,H],_):(_,N4),
{g(np(dependence(C1,C2)))}, {all(NA,NB)} ::>
X=C1, Y= C2, F1 \= F2-> assign(X, [F1,A,B,C,D],Y,[F2,E,F,G,H],
Rg, FRg, Rd, FRd);
X=C2, Y= C1, F1 \= F2 ->assign(Y,[F2,E,F,G,H],X,[F1,A,B,C,D],Rg,FRg,Rd,FRd) |
{fail(np(dependence(Rg, Rd)), FRg, FRd, N1, N4,NA,NB)}.
```

Figure 5.12: Dependency rules

Parsing with the obligatority constraint

Previous models of Womb Grammars failed if a noun was absent; the obligatority constraint now only fails if a phrase does not have a noun or pronoun or proper noun constituent as described in section 5.2.8. Obligatority succeeds if the converse occurs. This idea eliminates the possibility of obligatority failing because it contains a personal noun which is a valid obligatory category.

```
word(C1,F1,_):(N1,N2),{g(np(obligatority(C)))},
{all(A,B)}::> member(C1,C)|
{succeed(np(obligatority(C)),C1,F1,N1,N2,A,B)}.
```

Figure 5.13: Rule for succeeding obligatority

In *Figure 5.13*, we declare a list of categories *C* which contains a noun, proper-noun and pronoun (code implementation of the subset of property grammar of English noun phrases can be found in Appendix C). The obligatority constraint is satisfied if *C1* is a member of C.

This approach enables us to check if pronouns and proper nouns in Yorùbá perform the same head function as they do in English. Our results indicate that proper nouns and pronouns can indeed function as the head of a phrase as in English.

Parsing with precedence constraints

In parsing precedence constraint, we use the precedence rules of English. A precedence rule is satisfied if the target phrase has exactly the same ordering as English. For instance in Yorùbá, every occurrence of precedence(det, n), precedence(det, adj) fails while precedence(adj, n) fails in certain input phrases and succeeds in other input phrases (details of this in appendix 1). This gives us adequate information to make a valid conclusion that in Yorùbá determiners always come after nouns and after adjectives. However, with the presence of both failed and successful instances of adjectives and noun orderings, and a pattern of features responsible, we can only make conclusions that Yorùbá nouns and adjectives have different orderings. Our system materializes this in our output as:

```
-conditional precedence(adjective,noun):-
adjective precedes noun if adjective is plural,
quantity-uncountable, can be used for both animate and inanimate
nouns
```

Figure 5.14: A conditional precedence property showing one of the conditions for the ordering

Our results in *Figure 5.14* reiterate linguistic findings by Yorùbá grammarians [17, 20]. It is important we state that the conditional precedence our system shows is that which has identified unique features responsible for the order. So that, the other with more general features can be inferred but is not explicitly output by our model.

```
word(C1,F1,_):(N1, _),...,word(C2,F2,_):(_, N4),
{g(np(precedence(C1,C2)))}, {all(A, B)}::>
{succeed(np(precedence(C1,C2)), F1, F2, N1, N4, A, B)}.
```

Figure 5.15: Precedence rule for checking success

In *Figure 5.15* we show from our code that precedence constraints are satisified if and only if there exists a precedence(C1,C2) rule in English, and an input phrase where *C1* precedes textitC2. The categories *C1*, *C2* and the features *F1*, *F2* of the words are stored in a list if the precedence rule is satisfied. We also retrieve the word boundaries and phrase boundaries as described in 6.2.5.

Parsing with constituency constraints

A success driven approach is adapted for parsing constituency. This is because we are only interested in the presence of the constituent and not the number of times it occurs in parsed phrases. Previous Womb Grammar formalisms did not test for constituents and this resulted in adding constituents that are absent in the target grammar to the target grammar simply because they exist in the source grammar. We also have a function that can induce constituents that are in the target language but absent in the source. More of this in 5.2.6.

```
word(C1,F1,_):(N1, N2),{g(np(constituency(C1)))},
{all(A, B)}::>
{succeed(np(constituency(C1)), F1, N1, N2, A, B)}.
```

Figure 5.16: The constituency rule

In *Figure 5.16*, every time a word is found, the category of the word is tested with the constituency rule of English. If the category of the word is found in the English properties, then it is satisfied and added to a succeeded properties list and subsequently induced. We also store the word boundaries and phrase boundaries as in 6.2.5

Parsing with requirement constraints

We test requirement by checking the constituent that co-occur with another constituent in a phrase.

```
check_waits \ wait(Prop, F1, F2, N1, N2, A, B), g(Prop) <=>
fail(Prop, F1, F2, N1, N2, A, B).
check_waits <=> true.

word(Required,F1,_):(N1,N2),
{g(np(requirement(Required,Requiring)))},{all(A, B)} ::>
{wait(np(requirement(Required,Requiring)), F1, [], N1, N2, A, B)}.
{wait(np(requirement(_,Requiring)),_,_,_,_,_,_)},
!word(Requiring,_,_)<:>true.
```

Figure 5.17: Rules to test requirement failure

In *Figure 5.17*, we test whether requirement fails by checking that as daughter of the same mother a *Requiring* category does not occur with a *Required* category. We achieve this by saving the constraint *wait/7* everytime we find a word with a *Requiring* category and there exists a property that says *Requiring* category requires a *Required* category. We continue to wait till the entire phrase is parsed, if peradventure *Required* will be a category in the phrase which should not necessitate a failure. If no *Required* is found within that phrase, the category *Requiring* and its features *F2*, its word boundaries *(N1 and N2)* and the phrase boundaries *A,B* where *Requiring* was found are added to the failed properties list; if Required is found, we delete *wait/7*.

```
assign(A,B,C,D,A,B,C,D).

!word(X,FX,_):(N1,_N2),...,!word(Y,FY,_):(_,N4),
{g(np(requirement(C1,C2)))},{all(A,B)} <:>
X = C1, Y = C2 -> assign(X, FX, Y, FY, Requiring, FRg, Required, FRd);
X = C2, Y = C1 -> assign(Y,FY,X,FX,Requiring,FRg,Required,FRd) |
{succeed(np(requirement(Requiring, Required)), FRg, FRd, N1, N4,A,B)}.
```

Figure 5.18: The rule to test requirement success

In testing for success as in *Figure 5.18*, if a daughter *Requiring* of a phrase requires a sibling *Required*, we must check that in that phrase (e.g. np) when *Requiring* appears as a daughter, *Required* is also a daughter of that phrase. In the implementation, we check if a *Requiring* and *Required* occur in the same phrase. The order of the categories is not important here, as we are simply checking if it occurs anywhere within a phrase. We store the features *F1, F2* of the word to induce conditional properties in the second phase of parsing. We also store the word and phrase boundaries which were retrieved as in 6.2.5.

5.2.9 Parsing with properties not relevant to our subset of noun phrases

Parsing with exclusion constraints

In parsing exclusion, we are interested in finding that the categories in exclusion do not occur together in a phrase. For instance, np(exclusion(Noun, Adjective)) will succeed if phrases do not have both nouns and adjectives in the same phrase. A failure occurs if otherwise. We are not interested in the order of the categories, so that for instance, np(exclusion(Noun, Adjective)) and np(exclusion(Adjective, Noun)) should not occur in a phrase.

```
assign(A,B,C,D,A,B,C,D).

word(X,FX,_):(N1,_), ... , word(Y,FY,_):(_,N4),
{g(np(exclusion(C1,C2)))}, {all(A, B)}::>
X = C1, Y = C2 -> assign(X, FX, Y, FY, Rg, FRg, Rd, FRd);
X = C2, Y = C1 -> assign(Y,FY,X,FX,Rg,FRg,Rd,FRd) |
{fail(np(exclusion(Rg,Rd)),FRg,FRd, N1, N4, A, B)}.
```

Figure 5.19: Rule to test exclusion fail

In *Figure 5.19*, if a word with category X, features FX and word boundary *N1* occurs with a word with category Y, features FY and word boundary *N4* and there is a constraint that excludes categories *C1* and *C2*, and phrase boundaries A and B, we check if X and Y are equal to *C1* and *C2* or *C2* and *C1* respectively. We use the helper predicate *assign/8* because we are not interested in the order of categories. We save the exclusion property with the assigned categories *Rg* and *Rd*; features *FRg* and *FRd*; word boundaries *N1* and *N4* and phrase boundaries A and B.

```
check_wait \ waits(Prop, F1, N1, N2, A, B), g(Prop) <=>
fail(Prop, F1, N1, N2, A, B).
check_wait <=> true.

word(C1,F1,_):(N1,N2), {g(np(exclusion(C1,C2)))}, {all(A, B)}
::> {waits(np(exclusion(C1,C2)), F1, N1, N2, A, B)}.
{waits(np(exclusion(_,C2)),F2,N1,N2,A,B)}, {all(A, B)},
!word(C2,F2,_):(N1,N2)<:>true.

word(C2,F1,_):(N1,N2), {g(np(exclusion(C1,C2)))}, {all(A, B)}
::> {waits(np(exclusion(C1,C2)), F1, N1, N2, A, B)}.
{waits(np(exclusion(C1,_)),F1,N1,N2,A,B)}, {all(A, B)},
!word(C1,F1,_):(N1,N2)<:>true.
```

Figure 5.20: Rule to test exclusion success

In *figure 5.20*, we test whether exclusion succeeds by checking that if a category *C1* occurs, then it does not occur with a *C2* category and vice versa. We achieve this by saving the constraint *waits/6* everytime we find a word with a *C1* category and there exists a exclusion property says *C1* and *C2* category. We continue to wait till the entire phrase is parsed, if peradventure *C2* will be a category in the phrase which should not necessitate a success. If no *C2* is found within that phrase, the category *C1* and its features *F1*, its word boundaries *(N1 and N2)* and the phrase boundaries *A,B* where *C1* was found are added to the succeeded properties list; if *C2* is found, we delete it *waits/6*.

Parsing with unicity constraints

Unicity checks that a category is unique in a phrase. In *figure 5.21*, we check if a word category *C*, features *F1* and word boundaries *N1* occurs with another category *C*, features *F2*, word boundaries N4 and there is a constraint that *C* should be unique, and phrase boundaries *A* and *B*. If all those conditions are met, the unicity constraint is saved in the failed properties list together with the features *F1* and *F2*, word boundaries *N1* and *N2* and phrase boundaries *A* and *B*.

```
word(C,F1,_:(N1,_)), ... , word(C,F2,_):(_,N4),
{g(np(unicity(C)))}, {all(A, B)} ::>
{fail(np(unicity(C)), F1, F2, N1, N4, A, B)}.
```

Figure 5.21: Rule to test unicity failure

```
word(C,F1,_):(N1,N2),{g(np(unicity(C)))}, {all(A, B)}
::>{waits(np(unicity(C)),F1,N1,N2,A,B)}.
{waits(np(unicity(C)), F1, N1, N2, A, B)},{all(A, B)},
word(C,F1,_):(N1,N2)<:>true.
```

Figure 5.22: Rule to test unicity success

We test unicity success *Figure 5.22* like we test exclusion failure. However, in this case, we use *waits/6* to check if there is more than one category C. If a second category C is found, we simply delete it, success is no longer relevant.

5.3 Inducing the Grammar

Inducing the grammar occurs after the phrases have been parsed and tested with the relevant properties. We discuss how the different properties are induced and how conditional properties are inferred. Conditional properties are properties which only occur with categories which have certain specific features.

5.3.1 Inducing precedence properties

We have three rules for inducing precedence properties. In the first rule, every instance of that property succeeds, so it is induced as a part of the grammar. In the second rule, every instance of the property fails and the converse is induced. For example, if np(precedence(adjective, noun)) fails in all relevant input phrases, we induce

```
np(precedence(noun, adjectve)).
```

In all cases, we provide a count of relevant sentences the precedence rule occurs.

```
check3(Lists,[[Prop,_,Count]|Listf]):-
not(member([Prop,_,_], Lists)),
Prop = np(precedence(X,Y)),
not(member([np(precedence(Y,X)),_,_], Lists)),
not(member([np(precedence(X,Y)),_,_], Lists)),
write('-'),write(np(precedence(X,Y))), writeln(':-'),
write(X), write(' precedes '), write(Y),
write(' in all '), write(Count),
writeln(' relevant phrases'), nl,
check3(Lists, Listf).
```

Figure 5.23: The third rule for inducing precedence properties

5.3.2 Inducing obligatority properties

The obligatority property as mentioned previously is a list of categories which can function as the head of a phrase. In inducing obligatority, we check the failed properties list if there is any occurrence of a failure of one or more of the categories. If no failure is found, we check the number of categories that succeeded and output the number of phrases these categories were members of.

```
check4([[Prop,Features,Count]|Tailsucceed],Listfailed):-
not(member([Prop,_,_], Listfailed)),
Prop = np(obligatority(X)),
write('-'),write(np(obligatority(X))), writeln(':-'),
write(X),write(' are obligatority. It succeeds in all '),
write(Count), write(' occurrences'), writeln('.'),
check_trait(Features),
check4(Tailsucceed, Listfailed).
```

Figure 5.24: Rule for inducing obligatority

5.3.3 Inducing consitutency properties

Constituency constraints were only tested for success, thus every constituent of phrases is induced with a count of the times it occurs in the input phrases.

```
check4([[Prop,_,Count]|Tailsucceed],Listfailed):-
Prop = np(constituency(X)),
write('-'),write(X),
write(' is a constituent of noun phrases. It occurs in '),
write(Count), writeln(' phrases.'), nl,
check4(Tailsucceed, Listfailed).
```

Figure 5.25: Rule for inducing constituency

5.3.4 Inducing requirement properties

To induce requirement, we check the failed properties for an occurrence of a failed instance of the requirment property. If no failure is found, we induce the requirement property with the number of times the requirement occurred in relevant phrases.

```
check5([[Prop,_,Count]|Tailsucceed],Listfailed):-
not(member([Prop,_,_], Listf)),
Prop = np(requirement(X,Y)),
write('-'),write(np(requirement(X,Y))),
writeln(':-'), write(X),
write(' requires '), write(Y),
write(' in all '), write(Count), write(' phrases. '),
nl, nl,
check5(Tailsucceed, Listfailed).
```

Figure 5.26: Rule for inducing requirement

5.3.5 Inducing dependence properties

To induce dependence, we check the failed properties for an occurrence of a failed instance of the dependence property. If no failure is found, we induce the dependence property with the number of times the dependence occurred in relevant phrases.

```
check5([[Prop,_,Count]|Tailsucceed],Listfailed):-
not(member([Prop,_,_], Listfailed)),
Prop = np(dependence(X,Y)),
write('-'),write(np(dependence(X,Y))),
writeln(':-'), write(X),
write(' and '), write(Y),
write(' have a dependency relation '),
write(' in all '), write(Count), write(' phrases. '),
nl, nl,
check5(Tailsucceed, Listfailed).
```

Figure 5.27: Rule for inducing dependence

5.3.6 Inducing properties not relevant to our subset

Inducing exclusion properties

To induce exclusion, we check the failed properties for an occurrence of a failed instance of the exclusion property. If no failure is found, we induce the exclusion property with the number of times the exclusion occurred in relevant phrases.

```
check5([[Prop,_Traits,Count]|Tails],Listf):-
not(member([Prop,_,_], Listf)),
Prop = np(exclusion(X,Y)),
write('-'),write(np(exclusion(X,Y))),
writeln(':-'), write(X),write(' and '),write(Y),
write(' are in exclusion'),
write(' in all '), write(Count), write(' phrases. '),
nl, nl,
check5(Tails, Listf).
```

Figure 5.28: Rules to induce exclusion

Inducing unicity properties

To induce unicity, we check the failed properties for an occurrence of a failed instance of the unicity property. If no failure is found, we induce the unicity property with the number of times the property occurred in relevant phrases.

```
check5([[Prop,_Traits,Count]|Tails],Listf):-
not(member([Prop,_,_], Listf)),
Prop = np(unicity(X)),
write('-'),write(np(unicity(X))),
writeln(':-'), write(X),
write(' is unique '),
write(' in all '), write(Count), write(' phrases. '),
nl, nl,
check5(Tails, Listf).
```

Figure 5.29: Rules to induce unicity

5.3.7 How conditional properties are induced

The original Property-based model, as we have seen, succeeded in detecting and signalling mistakes in the sentence, without blocking the analysis. In this first model, which is parsing-oriented, incorrect sentences could be "accepted" through declaring some constraints as relaxable. The relaxable constraints are those which have some contexts in which they do not hold, and rules are written to indicate the contexts in which they should be relaxed. For instance, while from the context-free grammar rules shown in 4.3 we would not be able

to parse "the the book" (a common mistake from cutting and pasting in word processors), in the constraint-based formulation we can relax the uniqueness of determiner constraint.

Relaxation can be made conditional (e. g. a head noun's requirement for a determiner can be made relaxable in case the head noun is generic and in plural form, as in "Lions sleep tonight"). The failure of relaxable constraints is signalled in the output, but does not block the entire sentence's analysis. Implementations not including constraint relaxation capabilities implicitly consider all properties as relaxable. And in fact, when exploiting constraint-based parsing for grammar transformation rather than for parsing, this is exactly what we need, since in an unknown language any constraint may need to be "relaxed" and even of course, corrected. Constraint relaxation, however, has its own pitfalls in that it can get easily unwieldy, as more constraints are relaxed. This can result in a proliferation of relaxation conditions.

For that reason, we have considered it more appropriate, in the context of grammar induction, to state "exceptions" as part of a property itself, rather than separately in a relaxation definition. Thus we have created conditional properties, e.g. "conditional precedence(pronoun,noun)" which expresses that a pronoun preceding a noun is conditional. We give the condition, "if the pronoun is plural, is a personal pronoun and can be used in place of both animate and inanimate nouns". We define a property as conditional if it happens that there occurs a pattern for which the property fails for some phrases and succeeds for others. All constraints that fail and succeed are tested in this phase. We find these properties by searching for features which are unique to the failure and/or success of these constraints.The features which are responsible for this difference in behaviour forms the condition under which such property succeeds or fails.

Let us note that requirement(noun, determiner) is in fact a conditional property of English and the condition under which this requirement property will not hold is if the noun is plural and generic. So that while singular and / or definite forms of a noun require a determiner (e.g *The boy* and *The boys*), a plural generic noun does not require a determiner (e.g *Chickens lay eggs*).

We have also used conditional properties to express choice, e.g. whereas before we could state that one element was obligatory in a phrase, we can now state that a property needs to have one of a list of constituents. E.g., the g(obligatority([noun, pronoun, proper noun])) constraint only needs to find one of noun, pronoun or proper noun to succeed. If none of these is found, then a failure occurs which indicates that an obligatory head is absent.

Inducing conditional requirement properties

To induce conditional requirement, we are interested in finding what kind of *Requiring* category exists without a *Required* category. As a result, we check only the features of the first category of the requirement property in the failed properties list, with the first category of the requirment property in the succeeded properties list.

```
check1([[Prop,Traitshs,_]|Tails],
[[Prop,Traitshf,_]|Tailf]):-
Prop = np(requirement(X,Y)),
check_succeeded_traits([Trait,_,_], Traitshs),
check_failed_traits([Trait,_, _],Traitshf),
write('-'),write('Conditional '),write(Prop),
writeln(':-'), write(X),
write(' requires '), write(Y),
write(' if  '), write(X), write(' is  '),
write_trait(Trait), nl, nl,
check1(Tails,Tailf).
```

Inducing conditional dependence properties

To induce conditional dependence, we are interested in finding which features are responsible for instances of dependence. As a result, we compare the features of the first category of the dependence in the succeeded properties list with those of the first category of the dependence in the failed properties list and the features of the second category of the dependence of the succeeded properties list with the second category of the dependence of the failed properties list. If we find any unique features that occur in every instance of either a failure or a success, we use those features as the condition under which to induce the property. If no such feature is found, we check if there are any unique features that occur at least once.

Inducing Conditional Precedence

We have two rules for inducing conditional precedence. These rules are based on basic comparing the features of properties in the failed properties list with those in the succeeded properties list and a different approach may be adopted. The first rule compares each feature of the first category of the precedence property in the succeeded properties list, with each feature of the second category of the precedence property in the failed properties list and each feature of the second category of the precedence property in the succeeded properties list is compared with each feature of the first category of the precedence property in the failed properties list. The comparison is done to find any features that occur in every instance (of a failure or success) and that may be unique to the failure or succeess of the property. That feature will be the condition under which the precedence rule will be induced. The feature can be of only one category or of both. We do this because the order of categories of a precedence property in the failed properties list will be swapped in a succeeded properties list. For instance, given np(precedence(adjective,noun)), if this property fails, the features of the nouns (which will be the first category if the property fails) will be followed by those

```
check1([[Prop,Traitshs,_]|Tails],
[[Prop,Traitshf,_]|Tailf]):-
Prop = np(dependence(X,Y)),
check_succeeded_traits([Trait,_,_], Traitshs),
check_failed_traits([Trait,_, _],Traitshf),
write('-'),write('Conditional '),write(Prop),
writeln(':-'), write(X),
write(' and '), write(Y),
write(' have a dependency relation '),
write(' if  '), write(X), write(' is  '),
write_trait(Trait), nl, nl,
check1(Tails,Tailf);
Prop = np(dependence(X,Y)),
check_succeeded_traits2([_,Trait,_], Traitshs),
check_failed_traits2([_,Trait, _],Traitshf),
write('-'),write('Conditional '),write(Prop),
writeln(':-'), write(X),
write(' and '), write(Y),
write(' have a dependency relation '),
write(' if  '), write(Y), write(' is  '),
write_trait(Trait), nl, nl,
check1(Tails,Tailf).
```

of the adjective. In a succeed case, the features of the adjective will be followed by those of the noun.

```
write_features([]):-
write('.').
write_features([H|Features]):-
H = both,
write(' used for both animate and inanimate nouns, '),
write_features(Features).
write_features([H|Features]):-
H = n/a,
write_features(Features).
write_features([H|Features]):-
H \= n/a,
H \= both,
write(H),write(', '),
write_Features(Features).
```

Figure 5.30: Helper rules for conditional precedence rules

```
check_succeeded_features(_,[_]).
check_succeeded_features([H1, _, _],[[H1, _H2f, _]|T]):-
check_succeeded_features([H1, _, _], T).

check_succeeded_features2(_,[_]).
check_succeeded_features2([H1, H2, _],[[_, H2, _]|T]):-
check_succeeded_features2([H1, H2, _], T).

check_failed_features(_,[_]).
check_failed_features(H1,[[H1f, _H2f, _]|T]):-
H1 \= H1f,
check_failed_features(H1,T).

check_failed_features2(_,[_]).
check_failed_features2([H1,_,_],[[_H1f, H2f, _]|T]):-
H1 \= H2f,
check_failed_features2([H1,_,_],T).
```

```
check1([],[]).
check1(_,[]).
check1([],_).
check1([[Prop,[],_]|_Tailsucceed],[[Prop,_Traitshf,_]|_Tailfailed]).
check1([[Prop,Traitshs,_]|Tailsucceed],
[[Prop,Traitshf,_]|Tailf]):-
Prop = np(precedence(X,Y))->
check_succeeded_features([Features,_,_], Traitshs),
check_failed_features2([Features,_, _],Traitshf),
write('-'),write('Conditional '),write(Prop),
writeln(':-'), write(X),
write(' precedes '), write(Y),
write(' if  '), write(X), write(' is  '),
write_features(Features), nl, nl,
check1(Tailsucceed,Tailfailed);
Prop = np(precedence(X,Y))->
check_succeeded_features2([_,Features1,_], Traitshs),
check_failed_features([_,Features, _],Traitshf),
write('-'),write('Conditional '),write(Prop),
writeln(':-'), write(X),
write(' precedes '), write(Y),
write(' if  '), write(X), write(' is  '),
write_features(Features1), nl, nl,
check1(Tailsucceed,Tailfailed).
```

Figure 5.31: A Conditional Precedence Rule

In the second rule, we compare features in the order in which we did for the first, however, we are interested in finding if a feature exists at least once, even if it is not in every occurrence of a failure or success. The second rule will only be executed if the first rule does not. We are correct in adding this rule because, there may be more than one group of features responsible for the failure or success of a precedence property and this rule allows us to infer all of them, even if they occur only once in the relevant property.

5.3.8 Inducing properties not relevant to our subset of noun phrases

Inducing conditional unicity properties

Unicity is a unary property, therefore, to induce conditional unicity properties, we compare the features of the category of the unicity property in the failed properties list with those

of the succeeded properties list. If we find a specific features that occur in every succeeded or failed instance, we induce this property. Otherwise, we check if such a feature occurs at least once.

Inducing conditional exclusion properties

Conditional exclusion properties are induced just as dependence rules are induced.

5.4 Summary

We have describe how our model differs from previous implementations of womb grammars as well as the features of our model. Our model renders a gloss for each word in each input phrase, failed and succeeded properties are provided for each phrase, word boundaries and phrases boundaries within which each property occurs are also displayed. We also describe how we incorporate tones, save failed and / or succeeded properties, induce properties not included in the source language but present in the target language as well as how we handle syntactic number and gender.

We also explained how each property is parsed. We use a failure conscious and success conscious approach in order to be able to track differences between the source language and the target language. This approach has been useful in the induction of the grammar of the target language, especially in inducing conditional propeties.

Chapter 6

Results and supporting evidence of correctness

As we will show, our results have been consistent with linguistic research findings of Yorùbá grammar except our dependency results which we were not able to verify from literature. We used phrases generated by our CFG to evaluate our WG model and our induced grammar have shown similarities with the CFG. Details of the Induced grammar is in appendix A and the context-free grammar (CFG) of Yorùbá is in Appendix B.

It is important to state that despite equivalences that our induced grammar share with the CFG subset, our induced grammar explicitly encodes more information than the CFG. This is because, context-free formalisms are unable to directly accommodate extra information and their related linguistic constraints. This is because their grammar symbols can only be simple identifiers, with no possibility of carrying extra arguments, as logic grammar symbols can, to transmit and consult such extra information.

We summarize our results into constituency, precedence, requirement, dependency and obligatority.

1 Constituency: Our constituency results show that in Yorùbá, nouns, pronouns, proper-nouns, adjectives, quantifiers and determiners are allowable categories in noun phrases. These constituents are featured in noun phrases described by [73, 17, 20] as well as in our CFG.

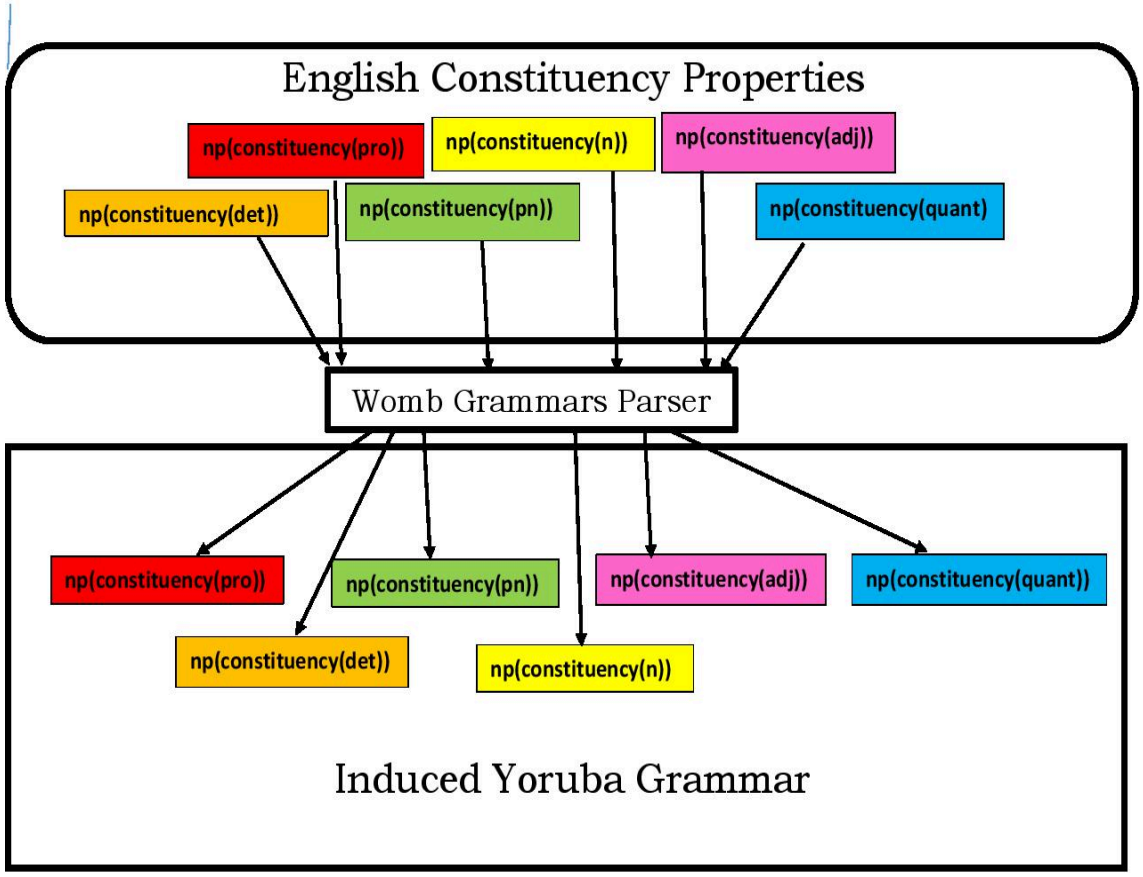


Figure 6.1: Summary of constituency results

In figure 6.1, the box labelled English constituency properties contains all the consitutency properties of English used in our model. We represent each constituent with different colours to enable readers to be able to easily read and identify the different constituents. The consituency properties are input into the womb grammar model, depicted with the use of arrows directed into the womb grammars box and the output is rendered in the box labelled as induced Yorùbá grammar. These properties are also labelled in different colours for the afore mentioned reasons. The box labelled *induce grammar of Yorùbá* thus shows only properties present in the target language.

2 Precedence: We induce three conditional precedence properties:

- Conditional np(precedence(pronoun,noun)):- pronoun precedes noun if pronoun is plural, neuter, used for both animate and inanimate nouns, third-person, [low,mid],
- Conditional np(precedence(adjective,noun)):- adjective precedes noun if

  adjective is plural, neuter, quantity-uncountable, used for both animate and inanimate nouns, [low-low-mid-low];

  adjective precedes noun if adjective is dual, neuter, attributive, animate, [mid-high-low],
- Conditional np(precedence(quantifier,noun)):- quantifier precedes noun if quantifier is plural, neuter, quantity-whole, used for both animate and inanimate nouns, [mid,mid]; and nine precedence properties detailed in appendix A.

Our conditional precedence imply that there are two orderings, which supports different orderings in pronouns and nouns, adjectives and nouns and in quantifier and nouns in for instance, rules 10 and 11; 14 and 15 and 13 and 17 of our CFG and also in literature [73].

We were also able to induce three precedence rules that are not in the English properties. We induced:

i np(precedence(proper-noun,adjective))

ii np(precedence(proper-noun,determiner))

iii np(precedence(quantifier,proper-noun))

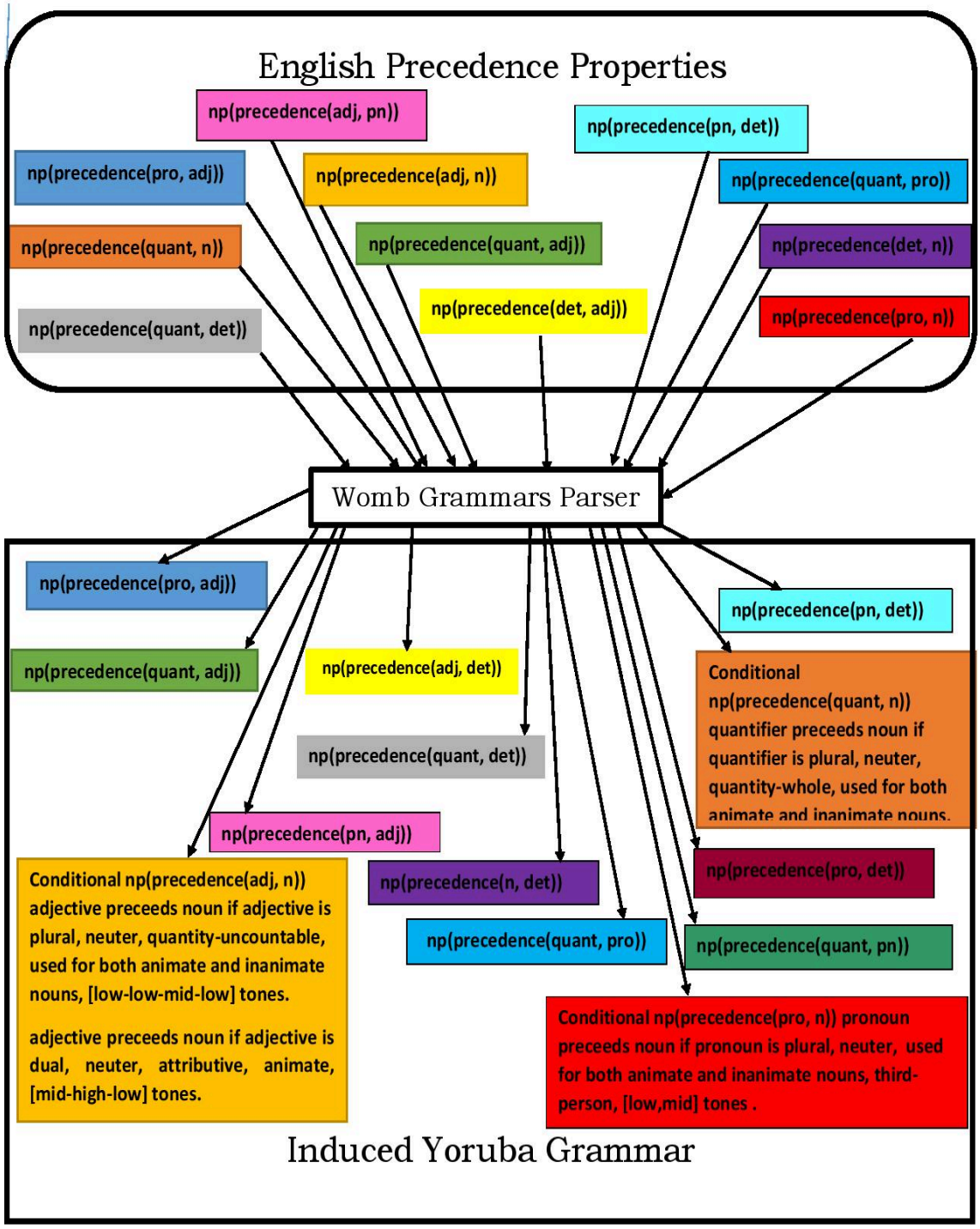


Figure 6.2: Summary of precedence results

In figure 6.2, the box labelled English precedence properties contains all the precedence properties of English used in our model. We represent each precedence constraint with different colours to enable readers to be able to easily read and identify the different constraints. The precedence properties are input into the womb grammar model, depicted with the use of arrows directed into the womb grammars box and the output is rendered in the box labelled as induced Yorùbá grammar. These properties are also labelled in different colours for the afore mentioned reasons. For conditional properties, we display only one ordering because, these are those induced by our model for having unique traits, and the converse orderings can be infered. We can assume that the ordering infered by our model is like an allomorph which occurs in specific contexts and the converse not inferred is like a morpheme which occurs in a more general context. The box labelled *induce grammar of Yorùbá* thus shows only properties present in the target language.

3 Requirement: We do not induce any requirement between nouns and determiner. This is because there exist no unique feature(s) which can be said to be responsible for the requirement property succeeding sometimes and failing at other times. This is consistent with our CFG which has rules where nouns occur without determiners as well as rules where nouns occur with determiners. This conclusion is also presented in research [20, 17, 10].

4 Dependency: Our model induced one dependency rule *np(dependency(quantifier, noun))*. Our system was able to find that dependency between quantifiers and nouns succeeded in all relevant phrases, but this is not explicit in our CFG. [10] lays claim to the fact that quantifiers have plural number and the nouns beside them take on their plural status, however, he did not explicitly mention a dependency relation between them.

5 Obligatority: Obligatority succeeded in all input phrases, showing that at least one of noun, pronoun and proper-nouns is a compulsory constituent of Yorùbá. Our CFG also shows these three constituents occur at least once in all rules.

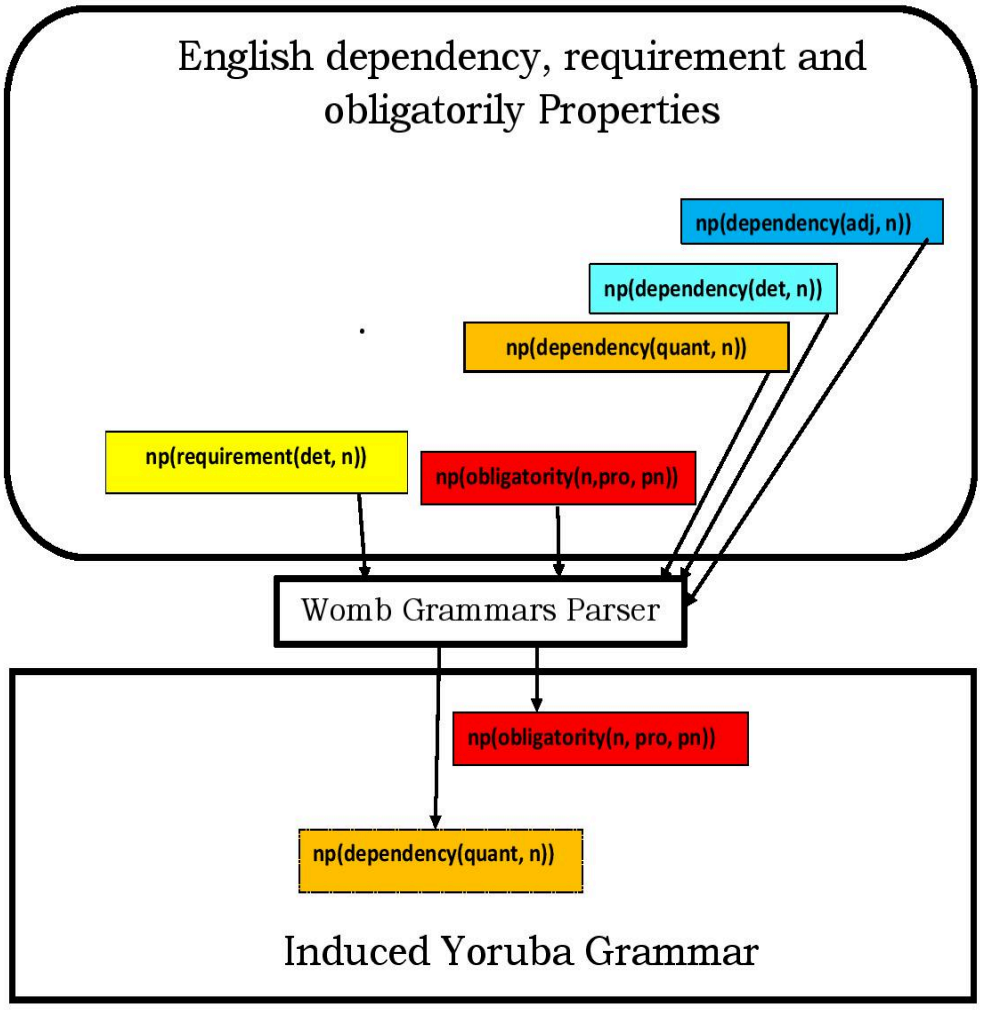


Figure 6.3: Summary of requirement, dependency and obligatority results

In figure 6.3, the box labelled English dependency, requirement and obligatority properties contains all the dependency, requirement and obligatority properties of English used in our model. We represent each dependency, requirement and obligatority constraint with different colours to enable readers to be able to easily read and identify the different constraints. The dependency, requirement and obligatority properties are input into the womb grammar model, depicted with the use of arrows directed into the womb grammars box and the output is rendered in the box labelled as induced Yorùbá grammar. These properties are also labelled in different colours for the afore mentioned reasons. The box labelled *induce grammar of Yorùbá* thus shows only properties present in the target language.

Chapter 7

Conclusion

We have shown the simplicity with which Womb Grammars automatically induce the grammar of Yorùbá from that of English despite the peculiarities in the grammar of Yorùbá and English that can make this very difficult which makes our model very useful in language development and language documentation.

We have also presented a concrete system, for inducing Yorùbá grammar that constitutes a proof of concept for the Womb Grammar model[42]. WGs we learnt is a system that allows users to input grammar description of a source language (ours being English) in modular and declarative fashion, in terms of the linguistic constraints that relate to the subset of noun phrases. We considered (constituency, precedence, dependency, obligatority, requirement). Since such descriptions also stand alone as a purely linguistic model of a language's syntax, our system can cater even for users who are not conversant with coding, such as pure linguists wanting to perform syntactic model transformation experiments aided by computers. Their purely (constraint-based) linguistic descriptions of syntax automatically turn into executable code when appended to our own code. We also used a subset of noun phrases which can accept any number of adjectives making our model extensible to induce even finer details such as precedence order of adjectives if we so desire.

As we have seen, our system automatically transforms a user's syntactic description of a source language into that of a target language, of which only the lexicon and a set of representative sample phrases are known. While demonstrated specifically for English as source language and Yorùbá as target language, our implementation can accept any other pair of languages for inducing the syntactic constraints of one from that of the other, as long as their description can be done in terms of the supported constraints.

We have thus adapted a flexible modelling framework solidly grounded in linguistic knowledge representation which, through constraint solving, turns into an efficiently executable problem solving tool. We maintain the modularity feature of the first implementation in our adaptation, therefore, our model can support revision and adaptation quite

straightforwardly: new constraints between immediate daughters of any phrase can be modularly added and implemented, without touching the rest of the system.

Our model allows for accommodating solution revisions at execution time by interacting with a linguist whenever constraints can not be induced consistently. Visual representations of the output that would be helpful for linguists in this respect have been explored in [5].

7.1 Possible extensions

We covered a useful subset of Yorùbá noun phrases, but there are some phenomena not covered here. For instance, there are some words that are noun phrases with a determiner interfixed between two words. For example, ọmọkọ́mọ (child any child), ilékílé (house any house), asọkásọ (dress any dress), ìwàkuwà (behaviour any behaviour), ìgbàkigbà (time any time) etc. In certain contexts, these words have a different meaning, as in, ọmọkọ̀mọ(child bad child), ilèkìlè(house bad house), ìwàkuwà (behaviour bad behaviour) etc and these contexts, determine the categories that can occur around them. It would be interesting to develop a parser that accomodates these features. Extending our program to identify and parse features such as vowel elision, assimilation of tones, compensatory lengthening and other phonological processes that have grammatical functions in Yorùbá would also be helpful.

Also, our formalism uses *word/3* to build its own lexicon manually, which makes it difficult to use a very large corpus. Subsequent versions will work at implementing a module that can automatically build the lexicon, perhaps using a tree bank. Further interesting ramifications of our work would also include: testing our system for a larger coverage of syntax (we have addressed noun phrases so far), inducing other constraints (besides precedence and constituency constraints which we induced) present in the source language but not in the target language, testing our model on other tonal-sensitive languages, studying how assimilation influences language structure especially in tone languages, studying whether any further constraints or approach extensions would be needed to accommodate families of languages not easily describable within our approach (e.g. languages which have different categories from the source language, those who have different inflectional paradigms from the source language, those that exhibit constraints not among our allowable set of constraints). Extending our work to also handle certain lexical, and syntactic ambiguities will be a step in the right direction.

Bibliography

[1] Aarts, B. (1998). Binominal Noun Phrases in English. Transactions of the Philological Society, 96: 117–158. doi: 10.1111/1467-968X.00025

[2] Abramson, H., Dahl, V.(1989). Logic Grammars. Monograph, Symbolic Computation AI Series. Springer-Verlag.

[3] Adebara, I.(2016). Using Womb Grammars for Inducing the Grammar of a Subset of Yorùbá Noun Phrases. In Triangle Journal, Tarragona.

[4] Adebara, I., Dahl, V. (2016). Grammar Induction as Automated Transformation between Constraint Solving Models of Language. In KNOWPROS, IJCAI 2016.

[5] Adebara, I., Dahl, V. (2015). Domes as a Prodigal Shape in Synthesis-Enhanced Parsers. In SHAPES 3.0, Constraints and Logic Programming (CSLP'04).

[6] Adebara, I., Dahl, V. (2015). Shape Analysis as an Aid for Grammar Inductions. In SHAPES 3.0, Constraints and Logic Programming (CSLP'04).

[7] Adebara, I., Dahl, V. (2015). Completing mixed language grammars through womb grammars plus ontologies. Henning Christiansen, M. D. J. L., and Loukanova, R., eds., Proceedings of the International Workshop on Partiality, Underspecification and Natural Language Processing, 32–40.

[8] Adegbija, E. (2004). Language Policy and Planning in Nigeria,Current Issues in Language Planning, 5:3, 181-246.

[9] Adeyanju, S., Adegbola, T., Fakinlede, O. (2015). A Supervised Phrase Selection Strategy for Phonetically Balanced Standard Yor'ub´a Corpus. In: Computational Linguistics and Intelligent Text Processing 16th International Conference, CICLing 2015, Cairo, Egypt, April 14-20, 2015, Proceedings, Part II. Springer International Publishing Switzerland 2015A. Gelbukh (Ed.), pp. 565582, 2015.

[10] Ajiboye, O.(2006). Interpreting Yor'ub´a Bare Nouns as Generic 31S Berkeley Linguistic Society Proceedings.

[11] Alishahi, A., Stevenson, S. (2008). A computational model of early argument structure acquisition. Cognitive Science, 32(5), 789–834.

[12] Alishahi A. (2011). Computational Modeling of Human Language Acquisition. Morgan and Claypool.

[13] Roberts, A., Atwell, E.(2002). Unsupervised Grammar Inference Systems for Natural Language.

[14] Angluin, D. Becerra-Bonach, L. (2008). Learning Meaning Before Syntax. In:Grammatical Inference: Algorithms and Applications: 9th International Colloquium, ICGI 2008 Saint-Malo, France, September 22-24, 2008 Proceedings

[15] Angluin, D. (1987). Learning regular sets from queries and counterexamples. Infrmation and Comp. 75, 87–106.

[16] Austin, P.K., Grenoble, L. (2007). Current Trends in Language Documentation. In Peter K. Austin (ed.) Language Documentation and Description, vol 4. London: SOAS. pp. 12-25

[17] Awobuluyi, O. (1978). Essentials of Yor'ub´a Grammar. Oxford University Press

[18] Awoyale, Y. (1988). Complex predicates and verb serialization. In: Lexicon Project Working Paper 28. Center for Cognitive Science, MIT, Cambridge, Mass.

[19] Awoyale Y. (2008). Global Yoruba Lexical Database v. 1.0 Linguistic Data Consortium, Philadelphia. Retrieved on 28/10/2015 from Natural Language Processing Lab, School of Computing Science, Simon Fraser University, British Columbia.

[20] Bamgbose, A.(1966). A Grammar of Yor'ub´a. Cambridge University Press.

[21] Bamgbose, A. (1993). Deprived, Endangered and Dying Languages. Diogenes March 1993 vol. 41 no. 161 19-25.

[22] Blache, P., Rauzy, S. (2012). Hybridization and Treebank Enrichment with Constraint-Based Representations. In: Proceedings of LREC.

[23] Blache, P. (2004). Property Grammars: A Fully Constraint-Based Theory. In:CSLP, pp. 1-16.

[24] Blache, P. (2000). Property grammars and the problem of constraint satisfaction. In: ESSLLI-2000 workshop on Linguistic Theory and Grammar Implementation.

[25] Blackburn, P., Bos, J. (2010). Computational Semantics. THEORIA: an International Journal for Theory. 18(1) pp27.

[26] Bird, S., Simons, G. (2003). Seven Dimensions of Portability for Language Documentation and Description. Language, Volume 79, Number 3, September 2003, pp. 557-582. Linguistic Society of America.

[27] Bowen, T.J. (1858). Grammar and Dictionary of the Yor'ub´a Language. Smithsonian Institution.

[28] Brill, E., Magerman, D. (1990). Marcus Met al. Deducing linguistic structure from the statistics of large corpus. In: Proceedings of the DARPA Speech and Natural Language Workshop, pp.275–282. Addison Wesley Longman Limited, Hidden Valley.

[29] Burkett D. and Klein D. (2008). Two Languages are Better than One (for Syntactic Parsing). In EMNLP. 877–886.

[30] Carrasco, R.C., Oncina, J. (1994). Learning Stochastic Regular Grammars by Means of a State Merging Method. Proceeding ICGI '94 Proceedings of the Second International Colloquium on Grammatical Inference and Applications Pages 139-152. Springer-Verlag London, UK

[31] Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In Proc. 14th Natl. Conf. Artif. Intell. pp. 598–603, AAAI Press.

[32] Chater, N., Vitanyi, P. (2007). Ideal learning of natural language: Positive results about learning from positive evidence. Journal of Mathematical Psychology, 51(3), 135–163.

[33] Chomsky, N. (1956). Three models for the description of language. I.R.E. Transactions on Information Theory. Proceedings of the symposium on information theory. Volumn IT-2, 113-123.

[34] Chomsky, N. (1957). Syntactic Structures. Walter de Gruyter.

[35] Christiansen, H. (2005). CHR grammars. TPLP, Number 4-5, Pgs 467-501.

[36] Clark A. (2004). Grammatical Inference and First Language Acquisition. In Workshop on Psycho-computational Models of Human Language Acquisition.

[37] Clark, A., Coste, F., Miclet, L. (2008). Grammatical Inference: Algorithms and Applications. 9th International Colloquium, ICGI 2008 Saint-Malo, France, September 22-24, 2008 Proceedings. Springer-Verlag Berlin Heidelberg.

[38] Cohen S.B. and Smith N.A. (2010). Covariance in Unsupervised Learning of Probabilistic Grammars. Journal of Machine Learning Research.11:3017–3051.

[39] Covington, M.A. (1994). Natural Language Processing for Prolog Programmers Prentice-Hall.

[40] Cramer, B. (2007). Limitations of current grammar induction algorithms. In ACL: Student Research.

[41] Cummins, C., Jan P. de Ruiter. (2014). Computational approaches o the pragmatics problem. Language and linguistics compass. Volume(8)(4) pp 133-143.

[42] Dahl, V., Miralles, E. (2012). Womb Parsing. In: 9th International Workshop on Constraint Handling Rules, Pp.32-40.

[43] Dahl, V., Miralles, J. (2012). Womb grammars: Constraint solving for grammar induction. In: Sneyers, J., Frühwirth, T. (eds.) Proceedings of the 9th Workshop on Constraint Handling Rules. vol. Technical Report CW 624, pp. 32–40. Department of Computer Science, K.U. Leuven.

[44] Dahl, V., Blache, P. (2004). Directly Executable Constraint Based Grammars. In: Proc. Journees Francophones de Programmation en Logique avec Contraintes.

[45] Dediu A., Martin-Vide C., Truthe B. (Eds.). (2013). Proceedings Language and Automata Theory and Applications. 7th International Conference, LATA 2013 Bilbao, Spain, April 2-5 2013, Springer-Verlag Berlin Heidelberg.

[46] de la Higuera, Colin. (2005). A bibliographical study of grammatical inference. Pattern Recognition 38: 1332–1348.

[47] Duchier, D., Dao, T.B.H., Parmentier,Y. (2013). Model-Theory and Implementation of Property Grammars with Features. In: Journal of Logic and Computation.

[48] Dupoint, P. (1994). Regular grammatical inference from positive and negative samples by genetic search: the GIG method. Second International Colloquium, ICGI-94 Alicante, Spain, September 21–23, 1994 Proceedings. pp 236-245.

[49] Edelman, S., Solan, Z., Ruppin, E., and Horn, D. (2004). Learning syntactic constructions from raw corpora. In Proceedings of the 29th Boston University Conference on Language Development, Boston:MA, USA.

[50] Fabunmi, F., Salawu, A. (2005). Is Yor'ub´a an Endangered Language? Journal of African Studies 14(3): 391408.

[51] Fagborun, O. (1994). The Yor'ub´a Koine - its history and linguistic innovations. Munchen: LINCOM EUROPA

[52] Fagborun, J. G. (1989). Some Practical Problems in Yoruba Orthography. In: The Journal of West African Languages. Vol (19)(2).

[53] Gold, E.M. (1967). Language identification in the limit. Information and Control, 10, pp. 447–474.

[54] Gippert, Jost, Himmelmann, N. P., Mosel, U. (eds.) (2006). Essentials of language documentation (Trends in Linguistics. Studies and Monographs, 178). Berlin: Mouton de Gruyter.

[55] Gliozzo, A., Strapparava, C.(2009). Semantic Domains in Computational Linguistics. Springer-Verlag Berlin Heidelberg.

[56] Glyn Morrill (2010). Categorial grammar: Logical syntax, semantics, and processing. Oxford University Press.

[57] Hana, J., Feldman, A.(2012). Resource-Light Approaches to Computational Morphology Part 1: Monolingual Approaches. Languages and linguistics compass. Volume(6)(10) pp 622-634.

[58] Heinz, J., Colin de la Higuera, Zaanen, M. (2016). Grammatical Inference for Computational Linguistics. Hirst, G. (Eds) Morgan and Claypool Publishers.

[59] Heinz, J., Sempere, J.M. (eds.). (2016). Topics in Grammatical Inference . Springer.

[60] Joshi, Aravind K., et al. (1975). Tree Adjunct Grammars. Journal of Computer Systems Science, Vol. 10 No. 1, pp. 136-163.

[61] Keizer, E(2007). The English noun phrase: the nature of linguistic categorization (Studies in English Language). Cambridge: Cambridge University Press.

[62] Keizer, M.E. (2002). Review of Inge de Mönnick, On the Move: the mobility of constituents in the English noun phrase: a multi-method approach. English Language and Linguistics 6/1, 210-216.

[63] Lari, K., Young, S.J. (1990). The estimation of stochastic context free grammars using the inside–outside algorithm. Comput. Speech Lang., 4 (1990), pp. 35–56

[64] Luca S.M. (2010). Grammatical Inference and Games: Extended Abstract. In: Grammatical inference: learning automata and grammars Colin de la Higuera (Eds). Cambridge ; New York: Cambridge University Press.

[65] Menno van Zaanen, Collin de la Higuera (2011). Computational Language Learning. In: Handbook of Logic and Language (Second Edition) edited by Johan van Benthem, Alice ter Meulen. Pp.765-780.,

[66] Merialdo, B. (1994). Tagging English text with a probabilistic model. Computational Linguistics 20(2), 129.

[67] Moore. E. (1956). Gedanken-experiments on sequential machines. Shannon, C. and McCarthy, J. (eds), Automata Studies, pages 129–153. Princeton University Press.

[68] Nicolas L. and Molinero M.A., Sagot B., Trigo E., de La Clergerie, Éric, Farré J. and Vergés J.M. (2009). Towards efficient production of linguistic resources: the Victoria Project.

[69] Nicholas, E. (2008). Review of Essentials of Language Documentation. Language Documentation and Conservation 2:340–350.

[70] Nikolaus, H. (1998). Documentary and descriptive linguistics. Linguistics 36(1):161–195.

[71] Odette, S.(2008). Fine-phonetic variation in a computational model of word recognition. The Journal of the Acoustical Society of America 123(5)

[72] Olumuyiwa, T. (2013). Yoruba Writing: Standards and Trends. In: Journal of Arts and Humanities, Vol(2)(1).

[73] Ogunbowale, T. (1970). Essentials of the Yoruba language. University of London Press.

[74] Parekh, R., Honavar, V. (2000). Automata induction, grammar inference, and language acquisition, in: H.L. Moise, R. Dale, H. Somers (Eds.), Handbook of Natural Language Processing, Marcel Dekker, New York.

[75] Perner, P. (Ed.). (2012). Machine Learning and Data Mining. In Pattern Recognition 8th International Conference, MLDM 2012 Berlin, Germany, July 13-20 Proceedings.

[76] Pollard, C., SAG, I. (1994). Head-Driven Phrase Structure Grammar. University of Chicago pres.

[77] Rasanen, O.(2011). A computational model of word segmentation from continuous speech using transitional probabilities of atomic acoustic events. Cognition Volume 120, Issue 2, Pp 149-176.

[78] Rathod, S.G. (2014). Machine translation of natural language using different approaches. International journal of computer applications 102(15):26-31.

[79] Romaine, S. (2002). The Impact of Language Policy on Endangered Languages. In: International Journal on Multicultural Societies, Vol. 4, No. 2,ISSN 1564-4901, UNESCO.

[80] Santorini, B. and Kroch, A. (2007). The syntax of natural language: An online introduction using the Trees program. http://www.ling.upenn.edu/ beatrice/syntax-textbook

[81] Searls, D.B. (2010). Molecules, Languages and Automata. In: Grammatical inference: learning automata and grammars. Colin de la Higuera(Eds). Cambridge ; New York: Cambridge University Press.

[82] Sempere, J.M., Garcia, P. (Eds). (2010). Grammatical Inference: Theoretical Results and Applications. In: 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Springer Berlin Heidelberg

[83] Shobhana, L. Chelliah, Willem J.de Reuse. (2010). Handbook of Descriptive Linguistic Fieldwork. Springer.

[84] http://www-01.sil.org/computing/catalog/index.asp?by=cat.

[85] Spears, W.M., Gordon, D.F. (2000). Evolution of strategies for resource protection problems. In: Advances in evolutionary computing: theory and applications, pp. 367–392. Springer, Heidelberg.

[86] Stevenson, A., Cordy, J.R. (2014). A survey of grammatical inference in software engineering. Sci. Comput. Program., 96, pp. 444–459

[87] Sun, M., Bellegarda, J.R. (2011). Improved Pos Tagging For Text-to-Speech Synthesis. In: ICASSP 2011, pp. 5384–5387.

[88] Thom W. Fr¨uhwirth, T.W. (1998). Theory and practice of constraint handling rules. J. Log. Program. 37(1-3), 95–138.

[89] Trudgill, P. (1986). Dialects in Contact. Oxford: Blackwell.

[90] Vamshi Ambati. 2012. Active Learning and Crowdsourcing for Machine Translation in Low Resource Scenarios. Ph.D. Dissertation. Carnegie Mellon Univ., Pittsburgh, PA, USA. Advisor(s) Jaime Carbonell. AAI3528171.

[91] Voutilainen, A. (2012). Part-of-Speech Tagging. The Oxford Handbook of Computational Linguistics Edited by Mitkov R.

[92] Wilson C. (2008). A maximum entropy model of phonotactics and phonotactic learning. Linguistic Inquiry 39. 379–440

[93] Woodbury, T. (2003). Defining documentary linguistics. In, Austin, P.(ed.), SOAS, 35-52.

[94] Wurm, S. (1998). Methods of Language Maintenance and Revival, with Selected Cases of Language Endangerment in the World. In: Studies in Endangered Languages, K. Matsumura (ed.). Tokyo: Hituzi Syobo.

[95] Y. Wang and T. Li (Eds.) (2011). Knowledge Engineering and Management, AISC 123, pp. 389– 396. Springer-Verlag Berlin Heidelberg.

[96] Yoon, J.W., Godsill, S.J., Kang, C., Kim, T-S. (2007). Bayesian inference for 2D gel electrophoresis image analysis, In Bioinformatics Research and Development, 343–356.

Appendix A

Results

1 <0> ọmọ <1>

```
child
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-1
Succeeded Property: np(constituency(noun))0-1; 0-1
Failed Property: np(requirement(noun,determiner))0-1; 0-1
```

2 <0> táyò <1>

```
Tayo
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-1
Succeeded Property: np(constituency(proper-noun))0-1; 0-1
```

3 <0> àwon <1>

```
they
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-1
Succeeded Property: np(constituency(pronoun))0-1; 0-1
```

4 <0> táyò <1> kan <2>

```
Tayo    a
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(determiner))1-2; 0-2
Succeeded Property: np(constituency(proper-noun))0-1; 0-2
Succeeded Property: np(precedence(proper-noun,determiner))0-2; 0-2
```

5 <0> táyò <1> kékeré <2> kan <3>

```
Tayo    small    a
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-3
Succeeded Property: np(constituency(determiner))2-3; 0-3
```

```
Succeeded Property: np(constituency(adjective))1-2; 0-3
Succeeded Property: np(constituency(proper-noun))0-1; 0-3
Failed Property: np(precedence(determiner,adjective))1-3; 0-3
Succeeded Property: np(precedence(proper-noun,determiner))0-3; 0-3
Succeeded Property: np(precedence(proper-noun,adjective))0-2; 0-3
```

6 <0> táyò <1> òbùn <2> kékeré <3> kan <4>

```
Tayo    dirty    small    a
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-4
Succeeded Property: np(constituency(determiner))3-4; 0-4
Succeeded Property: np(constituency(adjective))2-3; 0-4
Succeeded Property: np(constituency(adjective))1-2; 0-4
Succeeded Property: np(constituency(proper-noun))0-1; 0-4
Failed Property: np(precedence(determiner,adjective))2-4; 0-4
Failed Property: np(precedence(determiner,adjective))1-4; 0-4
Succeeded Property: np(precedence(proper-noun,determiner))0-4; 0-4
Succeeded Property: np(precedence(proper-noun,adjective))0-3; 0-4
Succeeded Property: np(precedence(proper-noun,adjective))0-2; 0-4
```

7 <0> akọ <1> eran <2> kékeré <3>

```
male    goat    small
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(constituency(adjective))2-3; 0-3
Succeeded Property: np(constituency(adjective))0-1; 0-3
Succeeded Property: np(constituency(noun))1-2; 0-3
Failed Property: np(precedence(adjective,noun))1-3; 0-3
Succeeded Property: np(precedence(adjective,noun))0-2; 0-3
Succeeded Property: np(dependence(adjective,noun))1-3; 0-3
Succeeded Property: np(dependence(adjective,noun))0-2; 0-3
Failed Property: np(requirement(noun,determiner))1-2; 0-3
```

8 <0> abo <1> eran <2> kékeré <3>

```
female    goat    small
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(constituency(adjective))2-3; 0-3
Succeeded Property: np(constituency(adjective))0-1; 0-3
Succeeded Property: np(constituency(noun))1-2; 0-3
Failed Property: np(precedence(adjective,noun))1-3; 0-3
Succeeded Property: np(precedence(adjective,noun))0-2; 0-3
Succeeded Property: np(dependence(adjective,noun))1-3; 0-3
Succeeded Property: np(dependence(adjective,noun))0-2; 0-3
Failed Property: np(requirement(noun,determiner))1-2; 0-3
```

9 <0> kékeré <1> eran <2>

```
small    goat
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(constituency(adjective))0-1; 0-2
Succeeded Property: np(constituency(noun))1-2; 0-2
Succeeded Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))1-2; 0-2
```

10 <0> aṣọ <1> kékeré <2> kan <3>

```
dress    small    a
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-3
Succeeded Property: np(constituency(determiner))2-3; 0-3
Succeeded Property: np(constituency(adjective))1-2; 0-3
Succeeded Property: np(constituency(noun))0-1; 0-3
Failed Property: np(precedence(determiner,adjective))1-3; 0-3
Failed Property: np(precedence(determiner,noun))0-3; 0-3
Failed Property: np(precedence(adjective,noun))0-2; 0-3
Succeeded Property: np(requirement(noun,determiner))0-3; 0-3
Failed Property: np(dependence(determiner,noun))0-3; 0-3
Succeeded Property: np(dependence(adjective,noun))0-2; 0-3
```

11 <0> aṣọ <1> funfun <2> kékeré <3>

```
dress    white    small
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-3
Succeeded Property: np(constituency(adjective))2-3; 0-3
Succeeded Property: np(constituency(adjective))1-2; 0-3
Succeeded Property: np(constituency(noun))0-1; 0-3
Failed Property: np(precedence(adjective,noun))0-3; 0-3
Failed Property: np(precedence(adjective,noun))0-2; 0-3
Succeeded Property: np(dependence(adjective,noun))0-3; 0-3
Succeeded Property: np(dependence(adjective,noun))0-2; 0-3
Failed Property: np(requirement(noun,determiner))0-1; 0-3
```

12 <0> eran <1> dúdú <2> ńlá <3> nàà <4>

```
goat    black    big    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-4
Succeeded Property: np(constituency(determiner))3-4; 0-4
Succeeded Property: np(constituency(adjective))2-3; 0-4
Succeeded Property: np(constituency(adjective))1-2; 0-4
Succeeded Property: np(constituency(noun))0-1; 0-4
Failed Property: np(precedence(determiner,adjective))2-4; 0-4
Failed Property: np(precedence(determiner,adjective))1-4; 0-4
Failed Property: np(precedence(determiner,noun))0-4; 0-4
Failed Property: np(precedence(adjective,noun))0-3; 0-4
```

```
Failed Property: np(precedence(adjective,noun))0-2; 0-4
Succeeded Property: np(requirement(noun,determiner))0-4; 0-4
Succeeded Property: np(dependence(determiner,noun))0-4; 0-4
Succeeded Property: np(dependence(adjective,noun))0-3; 0-4
Succeeded Property: np(dependence(adjective,noun))0-2; 0-4
```

13 <0> ìwo <1> náà <2>

```
you    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(determiner))1-2; 0-2
Succeeded Property: np(constituency(pronoun))0-1; 0-2
Succeeded Property: np(precedence(pronoun,determiner))0-2; 0-2
```

14 <0> ọmọ <1> náà <2>

```
child    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(determiner))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(determiner,noun))0-2; 0-2
Succeeded Property: np(requirement(noun,determiner))0-2; 0-2
Succeeded Property: np(dependence(determiner,noun))0-2; 0-2
```

15 <0> ọmọ <1> àkókó <2> náà <3>

```
child    first    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-3
Succeeded Property: np(constituency(determiner))2-3; 0-3
Succeeded Property: np(constituency(adjective))1-2; 0-3
Succeeded Property: np(constituency(noun))0-1; 0-3
Failed Property: np(precedence(determiner,adjective))1-3; 0-3
Failed Property: np(precedence(determiner,noun))0-3; 0-3
Failed Property: np(precedence(adjective,noun))0-2; 0-3
Succeeded Property: np(requirement(noun,determiner))0-3; 0-3
Succeeded Property: np(dependence(determiner,noun))0-3; 0-3
Succeeded Property: np(dependence(adjective,noun))0-2; 0-3
```

16 <0> àwon <1> ọmọ <2> náà <3>

```
they    child    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-3
Succeeded Property: np(constituency(determiner))2-3; 0-3
Succeeded Property: np(constituency(pronoun))0-1; 0-3
Succeeded Property: np(constituency(noun))1-2; 0-3
```

```
Failed Property: np(precedence(determiner,noun))1-3; 0-3
Succeeded Property: np(precedence(pronoun,noun))0-2; 0-3
Succeeded Property: np(requirement(noun,determiner))1-3; 0-3
Failed Property: np(dependence(determiner,noun))1-3; 0-3
Succeeded Property: np(precedence(pronoun,determiner))0-3; 0-3
```

17 <0> ajá <1> ńlá <2> kan <3>

```
dog    big    a
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-3
Succeeded Property: np(constituency(determiner))2-3; 0-3
Succeeded Property: np(constituency(adjective))1-2; 0-3
Succeeded Property: np(constituency(noun))0-1; 0-3
Failed Property: np(precedence(determiner,adjective))1-3; 0-3
Failed Property: np(precedence(determiner,noun))0-3; 0-3
Failed Property: np(precedence(adjective,noun))0-2; 0-3
Succeeded Property: np(requirement(noun,determiner))0-3; 0-3
Failed Property: np(dependence(determiner,noun))0-3; 0-3
Succeeded Property: np(dependence(adjective,noun))0-2; 0-3
```

18 <0> okùn <1> gígùn <2>

```
rope    long
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(adjective))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))0-1; 0-2
```

19 <0> okùn <1> gígùn <2> kan <3>

```
rope    long    a
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-3
Succeeded Property: np(constituency(determiner))2-3; 0-3
Succeeded Property: np(constituency(adjective))1-2; 0-3
Succeeded Property: np(constituency(noun))0-1; 0-3
Failed Property: np(precedence(determiner,adjective))1-3; 0-3
Failed Property: np(precedence(determiner,noun))0-3; 0-3
Failed Property: np(precedence(adjective,noun))0-2; 0-3
Succeeded Property: np(requirement(noun,determiner))0-3; 0-3
Failed Property: np(dependence(determiner,noun))0-3; 0-3
Succeeded Property: np(dependence(adjective,noun))0-2; 0-3
```

20 <0> ẹja <1> tútù <2>

```
fish    fresh
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(adjective))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))0-1; 0-2
```

21 <0> òkú <1> eran <2> kan <3>

```
dead    goat    a
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(constituency(determiner))2-3; 0-3
Succeeded Property: np(constituency(adjective))0-1; 0-3
Succeeded Property: np(constituency(noun))1-2; 0-3
Failed Property: np(precedence(determiner,adjective))0-3; 0-3
Failed Property: np(precedence(determiner,noun))1-3; 0-3
Succeeded Property: np(precedence(adjective,noun))0-2; 0-3
Succeeded Property: np(requirement(noun,determiner))1-3; 0-3
Succeeded Property: np(dependence(determiner,noun))1-3; 0-3
Failed Property: np(dependence(adjective,noun))0-2; 0-3
```

22 <0> ayé <1> kan <2>

```
life    a
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(determiner))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(determiner,noun))0-2; 0-2
Succeeded Property: np(requirement(noun,determiner))0-2; 0-2
Succeeded Property: np(dependence(determiner,noun))0-2; 0-2
```

23 <0> ilé-ìwé <1> náà <2>

```
school    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(determiner))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(determiner,noun))0-2; 0-2
Succeeded Property: np(requirement(noun,determiner))0-2; 0-2
Succeeded Property: np(dependence(determiner,noun))0-2; 0-2
```

24 <0> ìdajì <1> ọmọ <2> náà <3>

```
half    child    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
```

```
Succeeded Property: np(constituency(determiner))2-3; 0-3
Succeeded Property: np(constituency(quantifier))0-1; 0-3
Succeeded Property: np(constituency(noun))1-2; 0-3
Succeeded Property: np(precedence(quantifier,determiner))0-3; 0-3
Succeeded Property: np(precedence(quantifier,noun))0-2; 0-3
Failed Property: np(precedence(determiner,noun))1-3; 0-3
Succeeded Property: np(requirement(noun,determiner))1-3; 0-3
Failed Property: np(dependence(determiner,noun))1-3; 0-3
Succeeded Property: np(dependence(quantifier,noun))0-2; 0-3
```

25 <0> òbùn <1> ọmọ <2>

```
dirty    child
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(constituency(adjective))0-1; 0-2
Succeeded Property: np(constituency(noun))1-2; 0-2
Succeeded Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))1-2; 0-2
```

26 <0> ọmọ <1> òbùn <2>

```
child    dirty
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(adjective))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))0-1; 0-2
```

27 <0> àwon <1> ọmọ <2>

```
they    child
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(pronoun))0-1; 0-2
Succeeded Property: np(constituency(noun))1-2; 0-2
Succeeded Property: np(precedence(pronoun,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))1-2; 0-2
```

28 <0> àwon <1> ọmọ <2> púpò <3> náà <4>

```
they    child    plenty    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-4
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-4
Succeeded Property: np(constituency(determiner))3-4; 0-4
```

```
Succeeded Property: np(constituency(pronoun))0-1; 0-4
Succeeded Property: np(constituency(adjective))2-3; 0-4
Succeeded Property: np(constituency(noun))1-2; 0-4
Failed Property: np(precedence(determiner,adjective))2-4; 0-4
Failed Property: np(precedence(determiner,noun))1-4; 0-4
Succeeded Property: np(precedence(pronoun,noun))0-2; 0-4
Succeeded Property: np(precedence(pronoun,adjective))0-3; 0-4
Failed Property: np(precedence(adjective,noun))1-3; 0-4
Succeeded Property: np(requirement(noun,determiner))1-4; 0-4
Failed Property: np(dependence(determiner,noun))1-4; 0-4
Succeeded Property: np(dependence(adjective,noun))1-3; 0-4
Succeeded Property: np(precedence(pronoun,determiner))0-4; 0-4
```

29 <0> àwon <1> ọmọ <2> won <3> náà <4>

```
they    child    they    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))2-3; 0-4
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-4
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-4
Succeeded Property: np(constituency(determiner))3-4; 0-4
Succeeded Property: np(constituency(pronoun))2-3; 0-4
Succeeded Property: np(constituency(pronoun))0-1; 0-4
Succeeded Property: np(constituency(noun))1-2; 0-4
Failed Property: np(precedence(determiner,noun))1-4; 0-4
Failed Property: np(precedence(pronoun,noun))1-3; 0-4
Succeeded Property: np(precedence(pronoun,noun))0-2; 0-4
Succeeded Property: np(requirement(noun,determiner))1-4; 0-4
Failed Property: np(dependence(determiner,noun))1-4; 0-4
Succeeded Property: np(precedence(pronoun,determiner))2-4; 0-4
Succeeded Property: np(precedence(pronoun,determiner))0-4; 0-4
```

30 <0> àwon <1> ọmọ <2> púpò <3>

```
they    child    plenty
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-3
Succeeded Property: np(constituency(pronoun))0-1; 0-3
Succeeded Property: np(constituency(adjective))2-3; 0-3
Succeeded Property: np(constituency(noun))1-2; 0-3
Succeeded Property: np(precedence(pronoun,noun))0-2; 0-3
Succeeded Property: np(precedence(pronoun,adjective))0-3; 0-3
Failed Property: np(precedence(adjective,noun))1-3; 0-3
Succeeded Property: np(dependence(adjective,noun))1-3; 0-3
Failed Property: np(requirement(noun,determiner))1-2; 0-3
```

31 <0> gbogbo <1> aṣọ <2> kékeré <3>

```
all    dress    small
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(constituency(quantifier))0-1; 0-3
Succeeded Property: np(constituency(adjective))2-3; 0-3
Succeeded Property: np(constituency(noun))1-2; 0-3
Succeeded Property: np(precedence(quantifier,adjective))0-3; 0-3
Succeeded Property: np(precedence(quantifier,noun))0-2; 0-3
Failed Property: np(precedence(adjective,noun))1-3; 0-3
Succeeded Property: np(dependence(quantifier,noun))0-2; 0-3
Failed Property: np(dependence(adjective,noun))1-3; 0-3
Failed Property: np(requirement(noun,determiner))1-2; 0-3
```

32 <0> gbogbo <1> àwon <2> aṣọ <3> won <4> náà <5>

```
all    they    dress    they    the
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))3-4; 0-5
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))2-3; 0-5
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-5
Succeeded Property: np(constituency(determiner))4-5; 0-5
Succeeded Property: np(constituency(quantifier))0-1; 0-5
Succeeded Property: np(constituency(pronoun))3-4; 0-5
Succeeded Property: np(constituency(pronoun))1-2; 0-5
Succeeded Property: np(constituency(noun))2-3; 0-5
Succeeded Property: np(precedence(quantifier,pronoun))0-4; 0-5
Succeeded Property: np(precedence(quantifier,pronoun))0-2; 0-5
Succeeded Property: np(precedence(quantifier,determiner))0-5; 0-5
Succeeded Property: np(precedence(quantifier,noun))0-3; 0-5
Failed Property: np(precedence(determiner,noun))2-5; 0-5
Failed Property: np(precedence(pronoun,noun))2-4; 0-5
Succeeded Property: np(precedence(pronoun,noun))1-3; 0-5
Succeeded Property: np(requirement(noun,determiner))2-5; 0-5
Failed Property: np(dependence(determiner,noun))2-5; 0-5
Succeeded Property: np(dependence(quantifier,noun))0-3; 0-5
Succeeded Property: np(precedence(pronoun,determiner))3-5; 0-5
Succeeded Property: np(precedence(pronoun,determiner))1-5; 0-5
```

33 <0> gbogbo <1> àwon <2> aṣọ <3> funfun <4>

```
all    they    dress    white
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))2-3; 0-4
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-4
Succeeded Property: np(constituency(quantifier))0-1; 0-4
Succeeded Property: np(constituency(pronoun))1-2; 0-4
Succeeded Property: np(constituency(adjective))3-4; 0-4
Succeeded Property: np(constituency(noun))2-3; 0-4
Succeeded Property: np(precedence(quantifier,pronoun))0-2; 0-4
Succeeded Property: np(precedence(quantifier,adjective))0-4; 0-4
```

```
Succeeded Property: np(precedence(quantifier,noun))0-3; 0-4
Succeeded Property: np(precedence(pronoun,noun))1-3; 0-4
Succeeded Property: np(precedence(pronoun,adjective))1-4; 0-4
Failed Property: np(precedence(adjective,noun))2-4; 0-4
Succeeded Property: np(dependence(quantifier,noun))0-3; 0-4
Failed Property: np(dependence(adjective,noun))2-4; 0-4
Failed Property: np(requirement(noun,determiner))2-3; 0-4
```

34 <0> ìdajì <1> ọmọ <2> kékeré <3>

```
half    child    small
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(constituency(quantifier))0-1; 0-3
Succeeded Property: np(constituency(adjective))2-3; 0-3
Succeeded Property: np(constituency(noun))1-2; 0-3
Succeeded Property: np(precedence(quantifier,adjective))0-3; 0-3
Succeeded Property: np(precedence(quantifier,noun))0-2; 0-3
Failed Property: np(precedence(adjective,noun))1-3; 0-3
Succeeded Property: np(dependence(quantifier,noun))0-2; 0-3
Failed Property: np(dependence(adjective,noun))1-3; 0-3
Failed Property: np(requirement(noun,determiner))1-2; 0-3
```

35 <0> ìgbà <1> gbogbo <2>

```
time    all
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(quantifier))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(quantifier,noun))0-2; 0-2
Succeeded Property: np(dependence(quantifier,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))0-1; 0-2
```

36 <0> gbogbo <1> táyò <2>

```
all    Tayo
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(constituency(quantifier))0-1; 0-2
Succeeded Property: np(constituency(proper-noun))1-2; 0-2
Succeeded Property: np(precedence(quantifier,proper-noun))0-2; 0-2
```

37 <0> ìdajì <1> aṣọ <2>

```
half    dress
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(constituency(quantifier))0-1; 0-2
Succeeded Property: np(constituency(noun))1-2; 0-2
```

```
Succeeded Property: np(precedence(quantifier,noun))0-2; 0-2
Succeeded Property: np(dependence(quantifier,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))1-2; 0-2
```

38 <0> òpòlopò <1> ogbón <2>

```
plenty    wisdom
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(constituency(adjective))0-1; 0-2
Succeeded Property: np(constituency(noun))1-2; 0-2
Succeeded Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))1-2; 0-2
```

39 <0> ẹbí <1> gbogbo <2>

```
family    all
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(quantifier))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(quantifier,noun))0-2; 0-2
Succeeded Property: np(dependence(quantifier,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))0-1; 0-2
```

40 <0> gbogbo <1> àwon <2> ẹbí <3>

```
all    they    family
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))2-3; 0-3
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(constituency(quantifier))0-1; 0-3
Succeeded Property: np(constituency(pronoun))1-2; 0-3
Succeeded Property: np(constituency(noun))2-3; 0-3
Succeeded Property: np(precedence(quantifier,pronoun))0-2; 0-3
Succeeded Property: np(precedence(quantifier,noun))0-3; 0-3
Succeeded Property: np(precedence(pronoun,noun))1-3; 0-3
Succeeded Property: np(dependence(quantifier,noun))0-3; 0-3
Failed Property: np(requirement(noun,determiner))2-3; 0-3
```

41 <0> gbogbo <1> àwon <2> ilé-ìwé <3>

```
all    they    school
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))2-3; 0-3
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-3
Succeeded Property: np(constituency(quantifier))0-1; 0-3
Succeeded Property: np(constituency(pronoun))1-2; 0-3
Succeeded Property: np(constituency(noun))2-3; 0-3
```

```
Succeeded Property: np(precedence(quantifier,pronoun))0-2; 0-3
Succeeded Property: np(precedence(quantifier,noun))0-3; 0-3
Succeeded Property: np(precedence(pronoun,noun))1-3; 0-3
Succeeded Property: np(dependence(quantifier,noun))0-3; 0-3
Failed Property: np(requirement(noun,determiner))2-3; 0-3
```

42 <0> akúsè <1> ọmọ <2>

```
poor    child
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(constituency(adjective))0-1; 0-2
Succeeded Property: np(constituency(noun))1-2; 0-2
Succeeded Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))1-2; 0-2
```

43 <0> òpòlopò <1> èrò <2>

```
plenty    traveller
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(constituency(adjective))0-1; 0-2
Succeeded Property: np(constituency(noun))1-2; 0-2
Succeeded Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))1-2; 0-2
```

44 <0> òye <1> ńlá <2>

```
intelligence    big
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(adjective))1-2; 0-2
Succeeded Property: np(constituency(noun))0-1; 0-2
Failed Property: np(precedence(adjective,noun))0-2; 0-2
Succeeded Property: np(dependence(adjective,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))0-1; 0-2
```

45 <0> àwon <1> egbé <2>

```
they    group
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))1-2; 0-2
Succeeded Property: np(obligatority([noun,proper-noun,pronoun]))0-1; 0-2
Succeeded Property: np(constituency(pronoun))0-1; 0-2
Succeeded Property: np(constituency(noun))1-2; 0-2
Succeeded Property: np(precedence(pronoun,noun))0-2; 0-2
Failed Property: np(requirement(noun,determiner))1-2; 0-2
```

There are 45 phrases.

```
Failed properties with the number of times they fail:-
1[np(requirement(noun,determiner)),[[[plural,neuter,collective,animate,
[mid-high]],[],3], [[dual,neuter,abstract,inanimate,[mid-low]],[],1],
[[plural,neuter,collective,animate,[low-low]],[],1],
[[dual,neuter,common,animate,[mid-mid]],[],8],
[[plural,neuter,common,inanimate,[mid-high-low-high]],[],1],
[[plural,neuter,abstract,inanimate,[mid-high]],[],1],
[[plural,neuter,common,inanimate,[mid-mid]],[],3],
[[plural,neuter,abstract,inanimate,[low-low]],[],1],
[[plural,neuter,common,animate,[mid-mid]],[],3],
[[dual,neuter,common,inanimate,[mid-low]],[],1],
[[dual,neuter,common,inanimate,[mid-mid]],[],1]],24]
2[np(precedence(adjective,noun)),
[[[dual,neuter,abstract,inanimate,[mid-low]],
[dual,neuter,descriptive-size,both,[high-high]],1],
[[plural,neuter,common,animate,[mid-mid]],
[dual,neuter,descriptive-size,both,[high-mid-high]],1],
[[plural,neuter,common,inanimate,[mid-mid]],
[dual,neuter,descriptive-colour,both,[mid-mid]],1],
[[plural,neuter,common,inanimate,[mid-mid]],
[dual,neuter,descriptive-size,both,[high-mid-high]],1],
[[plural,neuter,common,animate,[mid-mid]],
[plural,neuter,quantity-countable,both,[high-low]],2],
[[dual,neuter,common,animate,[mid-mid]],
[dual,neuter,attributive,animate,[low-low]],1],
[[dual,neuter,common,animate,[mid-mid]],
[dual,neuter,descriptive,both,[high-low]],1],
[[dual,neuter,common,inanimate,[mid-low]],
[dual,neuter,descriptive,inanimate,[high-low]],2],
[[dual,neuter,common,animate,[mid-high]],
[dual,neuter,descriptive-size,both,[high-high]],1],
[[dual,neuter,common,animate,[mid-mid]],
[dual,neuter,number,both,[low-high-high]],1],
[[dual,neuter,common,animate,[mid-mid]],
[dual,neuter,descriptive-colour,both,[high-high]],1],
[[dual,neuter,common,animate,[mid-mid]],
[dual,neuter,descriptive-size,both,[high-high]],1],
[[dual,neuter,common,inanimate,[mid-mid]],
[dual,neuter,descriptive-colour,both,[mid-mid]],1],
[[dual,neuter,common,inanimate,[mid-mid]],
[dual,neuter,descriptive-size,both,[high-mid-high]],2],
[[dual,neuter,common,animate,[mid-mid]],
[dual,neuter,descriptive-size,both,[high-mid-high]],2]],19]
3[np(precedence(quantifier,noun)),[[[plural,neuter,collective,animate,
[mid-high]],
[plural,neuter,quantity-whole,both,[mid,mid]],1],
[[plural,neuter,abstract,inanimate,[low-low]],
```

```
[plural,neuter,quantity-whole,both,[mid,mid]],1]],2]
4[np(dependence(adjective,noun)),[[[dual,neuter,descriptive-size,both,
[high-mid-high]],[plural,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,descriptive-colour,both,[mid-mid]],
[plural,neuter,common,inanimate,[mid-mid]],1],
[[dual,neuter,descriptive-size,both,[high-mid-high]],
[plural,neuter,common,inanimate,[mid-mid]],1],
[[dual,neuter,attributive,animate,[low-high]],
[singular,neuter,common,animate,[mid-mid]],1]],4]
5[np(dependence(determiner,noun)),[[[dual,neuter,definite,both,[mid,low]],
[plural,neuter,common,inanimate,[mid-mid]],1],
[[dual,neuter,definite,both,[mid,low]],[plural,neuter,common,animate,
[mid-mid]],4],
[[singular,neuter,indefinite,both,[mid]],[dual,neuter,common,inanimate,
[mid-low]],1],
[[singular,neuter,indefinite,both,[mid]],[dual,neuter,common,animate,
[mid-high]],1],
[[singular,neuter,indefinite,both,[mid]],[dual,neuter,common,inanimate,
[mid-mid]],1]],8]
6[np(precedence(pronoun,noun)),[[[plural,neuter,common,inanimate,[mid-mid]],
[plural,neuter,both,third-person-elided,[mid]],1],
[[plural,neuter,common,animate,[mid-mid]],
[plural,neuter,both,third-person-elided,[mid]],1]],2]
7[np(precedence(determiner,noun)),[[[plural,neuter,common,inanimate,
[mid-mid]], [dual,neuter,definite,both,[mid,low]],1],
[[plural,neuter,common,animate,[mid-mid]],[dual,neuter,definite,both,
[mid,low]],4],
[[dual,neuter,common,inanimate,[mid-high-low-high]],
[dual,neuter,definite,both,[mid,low]],1],[[singular,neuter,abstract,
inanimate,[mid-high]],[singular,neuter,indefinite,both,[mid]],1],
[[singular,neuter,common,animate,[mid-mid]],[singular,neuter,
indefinite,both,[mid]],1], [[dual,neuter,common,inanimate,[mid-low]],
[singular,neuter,indefinite,both,[mid]],1],
[[dual,neuter,common,animate,[mid-high]],
[singular,neuter,indefinite,both,[mid]],1],
[[dual,neuter,common,animate,[mid-mid]],[dual,neuter,definite,both,
[mid,low]],3],[[dual,neuter,common,inanimate,[mid-mid]],
[singular,neuter,indefinite,both,[mid]],1]],14]
8[np(precedence(determiner,adjective)),
[[[plural,neuter,quantity-countable,both,[high-low]],
[dual,neuter,definite,both,[mid,low]],1],
[[dual,neuter,attributive,animate,[low-high]],
[singular,neuter,indefinite,both,[mid]],1],
[[dual,neuter,descriptive,inanimate,[high-low]],
[singular,neuter,indefinite,both,[mid]],1],
[[dual,neuter,descriptive-size,both,[high-high]],
[singular,neuter,indefinite,both,[mid]],1],
```

```
[[dual,neuter,number,both,[low-high-high]],
[dual,neuter,definite,both,[mid,low]],1],
[[dual,neuter,descriptive-colour,both,[high-high]],
[dual,neuter,definite,both,[mid,low]],1],
[[dual,neuter,descriptive-size,both,[high-high]],
[dual,neuter,definite,both,[mid,low]],1],
[[dual,neuter,descriptive-size,both,[high-mid-high]],
[singular,neuter,indefinite,both,[mid]],3],
[[dual,neuter,attributive,animate,[low-low]],
[singular,neuter,indefinite,both,[mid]],1]],11]

Succeeded properties with the number of times they succeed:-
1[np(precedence(pronoun,noun)),
[[[plural,neuter,both,third-person,[low,mid]],
[plural,neuter,collective,animate,[mid-high]],2],
[[plural,neuter,both,third-person,[low,mid]],
[plural,neuter,common,inanimate,[mid-high-low-high]],1],
[[plural,neuter,both,third-person,[low,mid]],
[plural,neuter,common,inanimate,[mid-mid]],2],
[[plural,neuter,both,third-person,[low,mid]],
[plural,neuter,common,animate,[mid-mid]],5]],10]
2[np(constituency(noun)),[[[plural,neuter,collective,animate,[mid-high]],3],
[[dual,neuter,abstract,inanimate,[mid-low]],1],
[[plural,neuter,collective,animate,[low-low]],1],
[[dual,neuter,common,animate,[mid-mid]],11],
[[plural,neuter,common,inanimate,[mid-high-low-high]],1],
[[plural,neuter,abstract,inanimate,[mid-high]],1],
[[plural,neuter,common,inanimate,[mid-mid]],4],
[[plural,neuter,abstract,inanimate,[low-low]],1],
[[plural,neuter,common,animate,[mid-mid]],7],
[[dual,neuter,common,inanimate,[mid-high-low-high]],1],
[[singular,neuter,abstract,inanimate,[mid-high]],1],
[[singular,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,common,inanimate,[mid-low]],2],
[[dual,neuter,common,animate,[mid-high]],1],
[[dual,neuter,common,inanimate,[mid-mid]],2]],38]
3[np(constituency(pronoun)),[[[plural,neuter,both,third-person,[low,mid]],11],
[[plural,neuter,both,third-person-elided,[mid]],2],
[[singular,neuter,both,second-person,[low,mid]],1]],14]
4[np(obligatority([noun,proper-noun,pronoun])),[[pronoun,14],[noun,38],
[proper-noun,5]],57]
5[np(dependence(adjective,noun)),
[[[dual,neuter,descriptive-size,both,[high-high]],
[dual,neuter,abstract,inanimate,[mid-low]],1],
[[plural,neuter,quantity-uncountable,both,[low-low-mid-low]],
[plural,neuter,collective,animate,[low-low]],1],
[[dual,neuter,attributive,animate,[mid-high-low]],
```

```
[dual,neuter,common,animate,[mid-mid]],1],
[[plural,neuter,quantity-uncountable,both,[low-low-mid-low]],
[plural,neuter,abstract,inanimate,[mid-high]],1],
[[plural,neuter,quantity-countable,both,[high-low]],
[plural,neuter,common,animate,[mid-mid]],2],
[[dual,neuter,attributive,animate,[low-low]],
[dual,neuter,common,animate,[mid-mid]],2],
[[dual,neuter,descriptive,both,[high-low]],
[dual,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,descriptive,inanimate,[high-low]],
[dual,neuter,common,inanimate,[mid-low]],2],
[[dual,neuter,descriptive-size,both,[high-high]],
[dual,neuter,common,animate,[mid-high]],1],
[[dual,neuter,number,both,[low-high-high]],
[dual,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,descriptive-colour,both,[high-high]],
[dual,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,descriptive-size,both,[high-high]],
[dual,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,descriptive-colour,both,[mid-mid]],
[dual,neuter,common,inanimate,[mid-mid]],1],
[[dual,neuter,descriptive-size,both,[high-mid-high]],
[dual,neuter,common,inanimate,[mid-mid]],2],
[[dual,neuter,descriptive-size,both,[high-mid-high]],
[dual,neuter,common,animate,[mid-mid]],3],
[[dual,feminine,attributive,animate,[mid-mid]],
[dual,neuter,common,animate,[mid-mid]],1],
[[dual,masculine,attributive,both,[mid-mid]],
[dual,neuter,common,animate,[mid-mid]],1]],23]
6[np(constituency(adjective)),[[[dual,neuter,descriptive-size,both,
[high-high]],3],
[[plural,neuter,quantity-uncountable,both,[low-low-mid-low]],2],
[[dual,neuter,attributive,animate,[mid-high-low]],1],
[[dual,neuter,descriptive-size,both,[high-mid-high]],9],
[[dual,neuter,descriptive-colour,both,[mid-mid]],2],
[[plural,neuter,quantity-countable,both,[high-low]],2],
[[dual,neuter,attributive,animate,[low-low]],3],
[[dual,neuter,attributive,animate,[low-high]],1],
[[dual,neuter,descriptive,both,[high-low]],1],
[[dual,neuter,descriptive,inanimate,[high-low]],2],
[[dual,neuter,number,both,[low-high-high]],1],
[[dual,neuter,descriptive-colour,both,[high-high]],1],
[[dual,feminine,attributive,animate,[mid-mid]],1],
[[dual,masculine,attributive,both,[mid-mid]],1]],30]
7[np(precedence(adjective,noun)),
[[[plural,neuter,quantity-uncountable,both,[low-low-mid-low]],
[plural,neuter,collective,animate,[low-low]],1],
```

```
[[dual,neuter,attributive,animate,[mid-high-low]],
[dual,neuter,common,animate,[mid-mid]],1],
[[plural,neuter,quantity-uncountable,both,[low-low-mid-low]],
[plural,neuter,abstract,inanimate,[mid-high]],1],
[[dual,neuter,attributive,animate,[low-low]],
[dual,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,attributive,animate,[low-high]],
[singular,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,descriptive-size,both,[high-mid-high]],
[dual,neuter,common,animate,[mid-mid]],1],
[[dual,feminine,attributive,animate,[mid-mid]],
[dual,neuter,common,animate,[mid-mid]],1],
[[dual,masculine,attributive,both,[mid-mid]],
[dual,neuter,common,animate,[mid-mid]],1]],8]
8[np(dependence(quantifier,noun)),[[[plural,neuter,quantity-whole,both,
[mid,mid]],[plural,neuter,common,inanimate,[mid-high-low-high]],1],
[[plural,neuter,quantity-whole,both,[mid,mid]],
[plural,neuter,collective,animate,[mid-high]],2],
[[plural,neuter,quantity-part,both,[low,mid,low]],
[plural,neuter,common,inanimate,[mid-mid]],1],
[[plural,neuter,quantity-whole,both,[mid,mid]],
[plural,neuter,abstract,inanimate,[low-low]],1],
[[plural,neuter,quantity-part,both,[low,mid,low]],
[plural,neuter,common,animate,[mid-mid]],2],
[[plural,neuter,quantity-whole,both,[mid,mid]],
[plural,neuter,common,inanimate,[mid-mid]],3]],10]
9[np(precedence(quantifier,noun)),[[[plural,neuter,quantity-whole,both,
[mid,mid]],[plural,neuter,common,inanimate,[mid-high-low-high]],1],
[[plural,neuter,quantity-whole,both,[mid,mid]],
[plural,neuter,collective,animate,[mid-high]],1],
[[plural,neuter,quantity-part,both,[low,mid,low]],
[plural,neuter,common,inanimate,[mid-mid]],1],
[[plural,neuter,quantity-part,both,[low,mid,low]],
[plural,neuter,common,animate,[mid-mid]],2],
[[plural,neuter,quantity-whole,both,[mid,mid]],
[plural,neuter,common,inanimate,[mid-mid]],3]],8]
10[np(precedence(quantifier,pronoun)),
[[[plural,neuter,quantity-whole,both,[mid,mid]],
[plural,neuter,both,third-person,[low,mid]],4],
[[plural,neuter,quantity-whole,both,[mid,mid]],
[plural,neuter,both,third-person-elided,[mid]],1]],5]
11[np(constituency(quantifier)),[[[plural,neuter,quantity-whole,both,
[mid,mid]],8],[[plural,neuter,quantity-part,both,[low,mid,low]],3]],11]
12[np(precedence(quantifier,proper-noun)),[[[plural,neuter,quantity-whole,
both,[mid,mid]],[dual,neuter,tba,n/a,[high,low]],1]],1]
13[np(constituency(proper-noun)),[[[dual,neuter,tba,n/a,[high,low]],5]],5]
14[np(precedence(quantifier,adjective)),[[[plural,neuter,quantity-part,both,
```

```
[low,mid,low]],[dual,neuter,descriptive-size,both,[high-mid-high]],1],
[[plural,neuter,quantity-whole,both,[mid,mid]],
[dual,neuter,descriptive-colour,both,[mid-mid]],1],
[[plural,neuter,quantity-whole,both,[mid,mid]],
[dual,neuter,descriptive-size,both,[high-mid-high]],1]],3]
15[np(precedence(pronoun,adjective)),[[[plural,neuter,both,third-person
,[low,mid]],[dual,neuter,descriptive-colour,both,[mid-mid]],1],
[[plural,neuter,both,third-person,[low,mid]],
[plural,neuter,quantity-countable,both,[high-low]],2]],3]
16[np(precedence(pronoun,determiner)),[[[plural,neuter,both,third-person,
[low,mid]],[dual,neuter,definite,both,[mid,low]],4],
[[plural,neuter,both,third-person-elided,[mid]],
[dual,neuter,definite,both,[mid,low]],2],
[[singular,neuter,both,second-person,[low,mid]],
[dual,neuter,definite,both,[mid,low]],1]],7]
17[np(requirement(noun,determiner)),[[[plural,neuter,common,inanimate,
[mid-mid]],[dual,neuter,definite,both,[mid,low]],1],
[[plural,neuter,common,animate,[mid-mid]],
[dual,neuter,definite,both,[mid,low]],4],
[[dual,neuter,common,inanimate,[mid-high-low-high]],
[dual,neuter,definite,both,[mid,low]],1],
[[singular,neuter,abstract,inanimate,[mid-high]],
[singular,neuter,indefinite,both,[mid]],1],
[[singular,neuter,common,animate,[mid-mid]],
[singular,neuter,indefinite,both,[mid]],1],
[[dual,neuter,common,inanimate,[mid-low]],
[singular,neuter,indefinite,both,[mid]],1],
[[dual,neuter,common,animate,[mid-high]],
[singular,neuter,indefinite,both,[mid]],1],
[[dual,neuter,common,animate,[mid-mid]],
[dual,neuter,definite,both,[mid,low]],3],
[[dual,neuter,common,inanimate,[mid-mid]],
[singular,neuter,indefinite,both,[mid]],1]],14]
18[np(precedence(quantifier,determiner)),
[[[plural,neuter,quantity-whole,both,[mid,mid]],
[dual,neuter,definite,both,[mid,low]],1],
[[plural,neuter,quantity-part,both,[low,mid,low]],
[dual,neuter,definite,both,[mid,low]],1]],2]
19[np(constituency(determiner)),[[[dual,neuter,definite,both,[mid,low]],10],
[[singular,neuter,indefinite,both,[mid]],8]],18]
20[np(dependence(determiner,noun)),[[[dual,neuter,definite,both,[mid,low]],
[dual,neuter,common,inanimate,[mid-high-low-high]],1],
[[singular,neuter,indefinite,both,[mid]],[singular,neuter,abstract,inanimate,
[mid-high]],1],[[singular,neuter,indefinite,both,[mid]],
[singular,neuter,common,animate,[mid-mid]],1],
[[dual,neuter,definite,both,[mid,low]],[dual,neuter,common,animate,
[mid-mid]],3]],6]
```

```
21[np(precedence(proper-noun,adjective)),[[[dual,neuter,tba,n/a,[high,low]],
[dual,neuter,attributive,animate,[low-low]],1],[[dual,neuter,tba,n/a,
[high,low]],[dual,neuter,descriptive-size,both,[high-mid-high]],2]],3]
22[np(precedence(proper-noun,determiner)),[[[dual,neuter,tba,n/a,[high,low]],
[singular,neuter,indefinite,both,[mid]],3]],3]


Induced Property Grammar rules for Yoruba Noun phrases::-

-Conditional np(precedence(pronoun,noun)):-
pronoun precedes noun if  pronoun is  plural, neuter,  used for both animate
 and inanimate nouns, third-person, [low,mid], .

-Conditional np(precedence(adjective,noun)):-
 adjective precedes noun if adjective is plural, neuter, quantity-uncountable,
  used for both animate and inanimate nouns, [low-low-mid-low], .

-Conditional np(precedence(adjective,noun)):-
 adjective precedes noun if adjective is dual, neuter, attributive, animate,
  [mid-high-low], .

-Conditional np(precedence(adjective,noun)):-
 adjective precedes noun if adjective is plural, neuter, quantity-uncountable,
  used for both animate and inanimate nouns, [low-low-mid-low].

-Conditional np(precedence(quantifier,noun)):-
 quantifier precedes noun if quantifier is plural, neuter, quantity-whole,
  used for both animate and inanimate nouns, [mid,mid], .

-np(precedence(quantifier,pronoun)):-
 quantifier precedes pronoun in all 5 relevant phrases


-np(precedence(quantifier,proper-noun)):-
 quantifier precedes proper-noun in all 1 relevant phrases


-np(precedence(quantifier,adjective)):-
 quantifier precedes adjective in all 3 relevant phrases


-np(precedence(pronoun,adjective)):-
 pronoun precedes adjective in all 3 relevant phrases


-np(precedence(pronoun,determiner)):-
pronoun precedes determiner in all 7 relevant phrases
```

```
-np(precedence(quantifier,determiner)):-
quantifier precedes determiner in all 2 relevant phrases


-np(precedence(proper-noun,adjective)):-
proper-noun precedes adjective in all 3 relevant phrases


-np(precedence(proper-noun,determiner)):-
proper-noun precedes determiner in all 3 relevant phrases


-np(precedence(determiner,noun)):-
determiner precedes noun in all 14 relevant phrases

-np(precedence(determiner,adjective)):-
determiner precedes adjective in all 11 relevant phrases

-noun is a constituent of noun phrases. It occurs in 38 phrases.

-pronoun is a constituent of noun phrases. It occurs in 14 phrases.

-np(obligatority([noun,proper-noun,pronoun])):-
[noun,proper-noun,pronoun] are obligatority.
It succeeds in all 57 occurrences.
pronoun succeeds in all 14 occurrences;

noun succeeds in all 38 occurrences;

proper-noun succeeds in all 5 occurrences;

-adjective is a constituent of noun phrases. It occurs in 30 phrases.

-quantifier is a constituent of noun phrases. It occurs in 11 phrases.

-proper-noun is a constituent of noun phrases. It occurs in 5 phrases.

-determiner is a constituent of noun phrases. It occurs in 18 phrases.

-np(dependence(quantifier,noun)):-
quantifier and noun have a dependency relation  in all 10 phrases.
```

Appendix B

Context free grammar of our subset Yorùbá noun phrases

We use *adj* for adjectives, *det* for determiners, *pro* for pronouns, *pn* for proper-nouns and *quant* for quantifiers. We also create variables that refer to the multiple categories that can occur in a particular environment. So that *adjs* refers to the presence of between 0 and two adjectives; *dets* refers to 0 or 1 determiner; *kernel* refers to nouns or proper-nouns. We accomodate any number of adjectives in our subset. [17].

```
1. adjs --> [].
2. adjs --> adj.
3. adjs --> adj, adjs.
4. dets--> det.
5. dets-->[].
6. kernel--> n.
7. kernel--> pn.
8. np--> kernel, adjs, dets.
9. np--> pro, dets.
10. np--> pro, n, adjs, dets.
11. np--> n, pro, dets.
12. np--> pro, n, pro, dets.
13. np--> n, quant.
14. np--> quant, n, adjs.
15. np--> adj, n.
16. np--> quant, kernel, dets.
17. np--> quant, pro, n, adj.
quant-->[gbogbo].%all
quant-->[idaji].%half
n-->[eran].%goat
n-->[iwe]. %book
n-->[ile-iwe]. %school
n-->[okun]. %rope
det-->[naa]. %definite article the
```

```
det-->[kan]. %indefinite article a
det-->[yii]. %this
adj-->[kekere].%small
adj-->[funfun].%white
adj-->[dudu].%black
adj-->[tutu].%fresh
adj-->[gigun].%long
adj-->[nla].%big
adj-->[pupo].%plenty
adj-->[obun].%big
adj-->[akoko].%big
pn-->[tayo].%proper noun
pro-->[awon]. % they
pro-->[tiwo]. % you
```

Appendix C

Property grammar of our subset of noun phrases of English

```
english_properties([
g(obligatority([noun,proper-noun,pronoun])),
g(constituency(determiner)),
g(constituency(quantifier)),
g(constituency(pronoun)),
g(constituency(adjective)),
g(constituency(noun)),
g(constituency(proper-noun)),
g(np(precedence(adjective, proper-noun))) %beautiful British Columbia
g(np(precedence(determiner, proper-noun))) %the United states
g(precedence(quantifier, pronoun)), %all his
g(precedence(quantifier, determiner)), %all the
g(precedence(quantifier, adjective)), %all beautiful
g(precedence(quantifier, noun)), %all books
g(precedence(determiner,adjective)), %the beautiful...
g(precedence(determiner,noun)),%the baby
g(precedence(pronoun,noun)), %Your sons e.g (all your sons)
g(precedence(pronoun,adjective)),%Your young e.g (all your young...)
g(precedence(adjective,noun)),%beautiful baby
g(requirement(noun, determiner)),
g(dependence(determiner,noun)),
g(dependence(quantifier,noun)),
g(dependence(adjective,noun))]).
```

Appendix D

Definitions

1 Turing machine is a quintiple (K, Σ, δ, s,H) where: K = Finite set of states Σ = an alphabet containing at least θ and $\Diamond$ but not $\leftarrow$ or $\rightarrow$ s $\in$ K is the start state, H $\subseteq$ K is the set of halting states, δ is a function such that: (K-H) * Σ to K * ($\Sigma \cup \rightarrow, \leftarrow$), if $\diamond$ is input symbol, then the action is $\rightarrow$ and $\diamond$ can never be written.

2 Recursively enumerable grammar Rules are of the form: $\alpha \rightarrow \beta$, where α and β are arbitrary strings over V and $\alpha \neq \epsilon$

3 A context sensitive grammar is a four tuple G =(N, Σ, P, S), where: N = Set of non-terminal symbols Σ = Set of terminal symbols S = Start symbol P = Finite set of productions rules. Rule are of the form: αA$\beta \rightarrow \alpha$Bβ S $\rightarrow \epsilon$ where A, S $\in$ N α, β, B $\in$ (N $\cup$ Σ)* B $\neq \epsilon$

4 A context-free grammar is a four tuple G =(V, Σ, P, S), where: V = A finite set of symbols called a vocabulary $\Sigma \subseteq$ V = Set of terminal symbols S $\in$ (V-Σ)= Start symbol P $\subseteq$(V-Σ)* V*= Finite set of productions rules. Rules are of the form: A $\rightarrow$ α where A $\in$ V, $\alpha \in$ (V $\cup$ Σ)

5 A regular grammar is a four tuple G =(N, Σ, P, S), where: N = A finite set of symbols called a vocabulary Σ = Set of terminal symbols S $\in$ N = Start symbol P = Finite set of productions, each having one of the following forms: A $\rightarrow$ aB A $\rightarrow$ a A $\rightarrow \epsilon$, where A, B, $\in$ N, a $\in \Sigma$ and ϵ is the empty string

Appendix E

Lexicon and input phrases in the Womb Grammar model

```
% LEXICON: word(Category,Traits,Word)
[kan]::> word(determiner,[singular,neuter, indefinite, both, [mid]],kan).
[náà]::> word(determiner,[dual,neuter, definite, both, [mid,low]],náà).
[yìí]::> word(determiner,[dual,neuter,demonstrative,both, [high,low]],yìí).

%nouns
[aṣọ]::> word(noun,[dual,neuter, common, inanimate, [mid-mid]],aṣọ).
[eran]::> word(noun,[dual,neuter, common, animate, [mid-mid]],eran).
[ọmọ]::>word(noun,[dual,neuter, common, animate, [mid-mid]],ọmọ).
[ẹja]::>word(noun,[dual,neuter, common, animate, [mid-mid]],ẹja).
[ajá]::>word(noun,[dual,neuter, common, animate, [mid-high]],ajá).
[okùn]::>word(noun,[dual,neuter, common, inanimate, [mid-low]],okùn).
[ilé-ìwé]::>word(noun,[dual,neuter,common,inanimate,
[mid-high-low-high]],ilé-ìwé).
[ìwé]::>word(noun,[dual,neuter, common, inanimate, [low-high]],ìwé).
[iṣé]::>word(noun,[dual,neuter, abstract, inanimate, [mid-high]],iṣé).
[ayé]::>word(noun,[dual,neuter, abstract, inanimate, [mid-high]],ayé).
[ìgbà]::>word(noun,[dual,neuter, abstract, inanimate, [low-low]],ìgbà).
[ogbón]::>word(noun,[dual,neuter, abstract, inanimate, [mid-high]],ogbón).
[òye]::>word(noun,[dual,neuter, abstract, inanimate, [mid-low]],òye).
[ẹbí]::>word(noun,[dual,neuter, collective, animate, [mid-high]],ẹbí).
[èrò]::>word(noun,[dual,neuter, collective, animate, [low-low]],èrò).
[egbé]::>word(noun,[dual,neuter, collective, animate, [mid-high]],egbé).

%adjectives
[kékeré] ::>
word(adjective,[dual,neuter,descriptive-size,both,[high-mid-high]],kékeré).
[ńlá] ::> word(adjective,[dual,neuter,descriptive-size,both, [high-high]],ńlá).
[dúdú] ::>
word(adjective,[dual,neuter, descriptive-colour, both,[high-high]],dúdú).
```

```
[funfun]::> word(adjective,[dual,neuter, descriptive-colour,both,
[mid-mid]],funfun).
[túùtù]::>word(adjective,[dual,neuter, descriptive, both, [high-low]],túùtù).
[gígùn]::>word(adjective,[dual,neuter,descriptive,inanimate,
[high-low]],gígùn).
[púpò]::>word(adjective,[plural,neuter, quantity-countable, both,
[high-low]],púpò).
[òpòlopò]::>word(adjective,[plural,neuter,quantity-uncountable, both,
[low-low-mid-low]],òpòlopò).
[àkókó]::> word(adjective,[dual,neuter, number, both, [low-high-high]],àkókó).
[òbùn]::>word(adjective,[dual,neuter, attributive, animate, [low-low]],òbùn).
[òkú] ::> word(adjective,[dual,neuter, attributive, animate, [low-high]],òkú).
[akúsè] ::> word(adjective,[dual,neuter, attributive, animate,
[mid-high-low]],akúsè).
[akọ]::> word(adjective,[dual,masculine,attributive,both,[mid-mid]],akọ).
[abo]::> word(adjective,[dual,feminine,attributive,animate,[mid-mid]],abo).


%quantifier
[gbogbo] ::> word(quantifier,[plural,neuter,quantity-whole,both,
[mid,mid]],gbogbo).
[ìdajì]::>word(quantifier,[plural,neuter,quantity-part,both,[low,mid,low]],ìdajì).

%proper noun
[táyò] ::> word(proper-noun,[dual,neuter,tba,n/a,[high,low]],táyò).

%pronoun
[àwon]::>word(pronoun,[plural,neuter,both,third-person,[low,mid]],àwon).
[ìwo]::>word(pronoun,[singular,neuter, both, second-person,[low,mid]],ìwo).
[tìwo]::>word(pronoun,[singular,neuter, both, second-person,[low,mid]],tìwo).
[won]::>word(pronoun,[plural,neuter, both, third-person-elided,[mid]],won).

tr(kan, a).
tr(nàà, the).
tr(yii, this).
tr(akọ, male).
tr(abo, female).
tr(aṣọ, dress).
tr(eran, goat).
tr(ọmọ, child).
tr(ẹja, fish).
tr(ajá, dog).
tr(okùn, rope).
tr(iṣé, work).
tr(ẹbí, family).
tr(ogbón, wisdom).
tr(òye, intelligence).
```

```
tr(ayé, life).
tr(ilé-ìwé, school).
tr(ìwé, book).
tr(ìgbà, time).
tr(èrò, traveller).
tr(egbé,group).
tr(kékeré, small).
tr(ńlá, big).
tr(dúdú, black).
tr(funfun, white).
tr(tútù, fresh).
tr(gígùn, long).
tr(púpò, plenty).
tr(òbùn, dirty).
tr(àkókó, first).
tr(òpòlopò, plenty).
tr(òkú, dead).
tr(akúsè, poor).
tr(gbogbo, all).
tr(ìdajì, half).
tr(táyò, 'Tayo').
tr(àwon, they).
tr(ìwo, you).
tr(tìwo, yours).
tr(won, they).


translate([]).
translate([Yoruba|Tail]):-
tr(Yoruba, English),%dictionary
write('   '),
write(English),
write(' '),
translate(Tail).

%SAMPLE PARSES/TESTS
inputstring([
[ọmọ],
[táyò],
[àwon],
[táyò, kan],
[táyò, kékeré, kan],
[táyò, òbùn, kékeré, kan],
[akọ, eran, kékeré],
[abo,eran, kékeré],
[kékeré, eran],
[aṣọ, kékeré, kan],
```

```
[aṣọ, funfun, kékeré],
[eran, dúdú, ńlá, náà],
[ìwo, náà],
[ọmọ,náà],
[ọmọ, àkókó, náà],
[àwon, ọmọ, náà],
[ajá, ńlá, kan],
[okùn, gígùn],
[okùn, gígùn, kan],
[ẹja, tútù],
[òkú, eran,kan],
[ayé, kan],
[ilé-ìwé, náà],
[ìdajì, ọmọ, náà],
[òbùn, ọmọ],
[ọmọ, òbùn],
[àwon, ọmọ],
[àwon, ọmọ, púpò, náà],
[àwon, ọmọ, won, náà],
[àwon, ọmọ, púpò],
[gbogbo, aṣọ, kékeré],
[gbogbo, àwon, aṣọ, won, náà],
[gbogbo, àwon, aṣọ, funfun],
[ìdajì, ọmọ, kékeré],
[ìgbà, gbogbo],
[gbogbo,táyò],
[ìdajì, aṣọ],
[òpòlopò, ogbón],
[ẹbí,gbogbo],
[gbogbo, àwon, ẹbí],
[gbogbo, àwon, ilé-ìwé ],
[akúsè, ọmọ],
[òpòlopò,èrò],
[òye, ńlá],
[àwon, egbé]
]).
```