A DEEP SYNTACTIC PARSER FOR THE TAMIL LANGUAGE

Kengatharaiyer Sarveswaran

178097E

Degree of Doctor of Philosophy

Department of Computer Science & Engineering

University of Moratuwa
Sri Lanka

September 2022

A DEEP SYNTACTIC PARSER FOR THE TAMIL LANGUAGE

Kengatharaiyer Sarveswaran

178097E

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Department of Computer Science & Engineering

University of Moratuwa
Sri Lanka

September 2022

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: UOM Verified Signature Date: 20/09/2022

The above candidate has carried out research for the PhD thesis under our supervision.

Name: Professor Gihan Dias, University of Moratuwa, Sri Lanka.

Signature of the Supervisor: UOM Verified Signature

Name: Professor Miriam Butt, University of Konstanz, Germany.

Signature of the Supervisor: UOM Verified Signature Date: 20/09/2022

ABSTRACT

A Deep Syntactic Parser for the Tamil Language

Natural Language Processing (NLP) applications have become integral to human life. A syntactic parser is a vital linguistic tool that shows syntactic relations between the words in a sentence. These may then be mapped to a tree, a graph, or a formal structure. Syntactic parsers are helpful for building other NLP applications. In addition, they help linguists to understand a language better and perform cross-lingual linguistic analysis. A syntactic parser that performs a deeper analysis and captures argumentative, attributive and coordinative relations between the words of a given sentence is called a deep syntactic parser. Tamil is considered a low-resourced language in terms of tools, applications, and resources available for others to use and build NLP applications or carry out linguistic analyses. Not many resources, such as treebanks and annotated corpora, or linguistic analysis tools such as POS taggers or morphological analysers, are publicly available for Tamil. Available off-the-shelf language-agnostic syntactic parsers show comparatively low performance because of the rich morphosyntactic properties of Tamil. This study elaborates on how I developed the first grammar-driven parser for Tamil, which uses the Lexical-Functional Grammar formalism, and a state-of-the-art data-driven parser using the Universal Dependencies framework. I have also proposed an approach to evaluate a syntactic parser's syntactical coverage, experimented with transition-based and graph-based approaches, and for the first time, tried multi-lingual training to develop a data-driven parser for Tamil. A part of speech tagger, a morphological analyser cum generator, pre-processing tools, and treebanks are the other tools and resources I have developed to facilitate the development of the parsers. While all these tools give the current best score for their respective tasks, these resources are also available online for others to build upon. Moreover, the study also documents my contributions toward understanding different linguistic aspects of the Tamil language.

Keywords: Deep Syntactic Parser; Grammar-driven parser; Data-driven parser; Part of Speech tagger; Morphological Analyser

DEDICATION

அப்பா - அம்மா

appā - ammā

‘Father - Mother’

for their unconditional love, support, and being the reason of who I am today.

பெரியப்பா - பெரியம்மா

periyappā - periyammā

‘Uncle - Aunt’

for being my guardians when crossing the most important part of my life.

இயற்கை

*iya*r̲*kai*

‘the great Nature’

(the god)

for always putting together and aligning things I required for the progressions of this study.

ACKNOWLEDGEMENTS

I am very grateful to the people who have supported me in various ways since the beginning of this study in 2018 to complete this study.

First, I must thank my two perfect supervisors, Professor Gihan Dias and Professor Miriam Butt. I would not have completed this study without their tremendous support. Their detailed and insightful comments have significantly improved my research work. I am additionally thankful to them for creating valuable opportunities to widen my knowledge and academic network.

I thank Professor Gihan Dias for encouraging me to start my PhD research and offering me the wonderful opportunity to be his doctoral student. He always knew the right moment when to lend me a word of encouragement and give me a push. I am grateful for that and for the valuable guidance he has provided to complete this study. Professor Gihan Dias also supported me with the funding to carry out research, attend conferences, and organise academic events.

I thank Professor Miriam Butt, my other supervisor, for helping me with the computational linguistics part of my study. She hosted me twice as a visiting researcher at the University of Konstanz and created opportunities to meet other leading scholars in this field of study. Professor Miriam also provided me with financial support to be able to attend conferences and supported me in co-organising workshops and a summer school. I am grateful for all these and the trust she kept in me.

I want to express my gratitude to the progress evaluation panel members, Professor Subathini Ramesh and Dr Charith Chitraranjan, and research coordinators Professor Sanath Jeyasena and Dr Shehan Fernando, for their continuous input and guidance in tailoring my research.

I am grateful to the following thesis evaluation panel members for their constructive comments and feedback to improve the thesis and plan my future work:

- Professor Jagath Premachandra - Professor at the University of Moratuwa, Sri Lanka.
- Professor Mary Dalrymple - Professor of Syntax at the University of Oxford, United Kingdom.
- Professor Sarmad Hussain - Professor of Computer Science, University of Engineering and Technology, Pakistan.

- Dr Uthayasanker Thayasivam - Senior lecturer in Computer Science at the University of Moratuwa, Sri Lanka.
- Dr Amal Shehan Perera - Senior lecturer in Computer Science at the University of Moratuwa, Sri Lanka.

I should thank my friend Dr Maris Camilleri for helping me with language editing and providing feedback on my write-up. I know, at times, she sacrificed her personal and family time to do these. I am very grateful to her in this respect.

I would like to thank my colleagues at the Department of Computer Science, the University of Jaffna for encouraging me to start the PhD research, and the administration of the University of Jaffna for providing me with the required permission and leave to carry out the research.

I am also thankful to the wonderful staff and researchers of the National Languages Processing centre, and the head and the staff of the Department of Computer Science and Engineering at the University of Moratuwa for their support, input, and the provision of all required resources. Moreover, I am grateful to them for making my time at the University of Moratuwa memorable and valuable. I would also like to thank the director and the staff Faculty of Graduate Studies for facilitating all administrative matters related to the registration and evaluation of my study.

I also thank my friends in the Computational Linguistics group at the University of Konstanz for making my stay at Konstanz productive and memorable. I would especially like to thank Jessica Zipf for her support in solving issues related to XLE.

I am thankful to Paul Meurer at the University of Bergen for providing the required permission and helping to set up the Tamil grammar on INESS. I am grateful to Dr Dan Zeman, Charles University, the Czech Republic, for helping me sort out technical issues and set up a Tamil Universal Dependencies treebank.

I would like to thank Professor Lauri Karttunen from Stanford University for introducing me Foma (a Finite-State compiler), which cleared one of my greatest obstacles in the development of the morphological analyser, and Dr Mans Hulden from the University of Colorado Boulder, for helping me sort out issues related to Foma.

I thank Dr Parameswari Krishnamurthy from the University of Hyderabad, for working with me to create a Tamil treebank that I then used as a part of my test set. I also would like to thank Professor Rajendran Sankaravelayuthan, Amrita Vishwa Vidyapeetham, Dr S. Rajamathangi, Jawaharlal Nehru University, and Ms Sujiththa Srikantharajah, University of Jaffna for helping me with reviewing linguistic annotations.

I owe a lot to my wife, Sharanyaa and daughters, Thenmozhi and Poongkuzhalii, for their patience and love. It would have been much easier for them if I had continued working at the University of Jaffna, but they encouraged me to go ahead with my studies. Our little daughter Nilavezhilii is the best thing that happened to us during

this challenging time. I know they made sacrifices to help me stay focused on my studies. I also thank my parents and brothers for supporting me in everything with selfless generosity.

காலத்தி னாற்செய்த நன்றி சிறிதெனினும்
ஞாலத்தின் மாணப் பெரிது. - திருக்குறள் (102)

kālatti n̲ār̲ceyta nan̲r̲i ci̲r̲iten̲in̲um
ñālattin̲ māṇap peritu. - tirukku̲ra̲ḷ (102)

“A favour conferred in the time of need,
though it be small (in itself),
is (in value) much larger than the world.”

Thank you!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
1S	First person – Singular
1PLE	First person - Plural - Epicene
3SN	Third person – Singular – Neuter
3SM	Third person – Singular – Masculine
3SF	Third person – Singular – Feminine
ABL	Ablative
ACC	Accusative
ADJ	Adjective
ADJL	Adjectivalizer
ADV	Adverb
ADVL	Adverbializer
AGR	Agreement
AP	Adjectival Participle
ARG	Argument
ASCII	American Standard Code for Information Interchange
ASS	Associative
AUX	Auxiliary
AVM	Attribute-Value-Matrix
BEN	Benefactive
BILSTM	Bidirectional Long Short-Term Memory
BLEX	Bi-LEXical dependency score
CAR	Cardinal
CAUS	Cause Marker
COMP	Complementiser
COND	Conditional
CONJ	Conjunction
COP	Copula
CRF	Conditional Random Field

DAT	Dative
DEC	Declarative
DIST	Distal
DOM	Differential Object Marking
DUR	Durative
EMPH	Emphatic
EPI	Epicene
F	Feminine
FEATS	Features
FST	Finite-State Transducer
FUT	Future Tense
GEN	Genitive
GEND	Gender
HDT	Hamburg Dependency Treebank
HON	Honorific
HON	Double Honorific
HORT	Hortative
IMP	Imperative
INCL	Inclusive
INESS	Infrastructure for the Exploration of Syntax and Semantics
INF	Infinitive
INS	Instrumental
IRRAT	Irrational
INTJ	Interjection
IOBJ	Indirect Object
LOC	Locative
LAS	Labelled Attachment Score
LFG	Lexical Functional Gramma
LV	Light Verb
M	Masculine Analyzer
MA	Morphological Azad
MLAS	Morphology-aware Labeled Attachment Score
MWTT	Modern Written Tamil Treebank
N	Neuter
NEG	Negative
NER	Named Entity Recognizer
NLP	Natural Language Processing
NLTK	Natural Language ToolKit
NMLZ	Nominaliser

NOM	Nominative
NP	Noun Phrase
NTYPE	Noun Type
NUM	Number
OBJ	Object
OBL	Oblique
ORD	Ordinal
PART	Particle
PASS	Passive
PERM	Permissive
PERS	Person
PL	Plural
POS	Parts of Speech
POSS	Possessive
PP	Postposition Phrase
PRED	Predicate
PRES	Present tense
PROG	Progressive
PRON	Pronoun
PRS	Present Tense
PSP	Postposition
PST	Past Tense
QUOT	Quotative
RAT	Rational
REL	Relativiser
RNN	Rrecurrent Neural Network
SAN	Sandhi
SEM	Semantic
SER	Singular - Epicene - Rational
SG	Singular
SUBJ	Subject
SVC	Serial Verb Construction
SYM	Symbol
TB	Treebank
TNS-ASP	Tense-Aspect
TTB	Tamil TreeBank
UAS	Unlabelled Attachment Score
UD	The Universal Dependencies
UPOS	Universal Part of Speech

VP	Verbal Pharse
VPART	Adverbial Participle
VTYPE	Verb type
XCOMP	Non-finite clause argument
XLE	Xerox Linguistic Engine
XPOS	Language-specific Part of Speech

TRANSLITERATION SCHEMA

Vowels		Consonants	
அ	a	க்	k
ஆ	ā	ங்	ṅ
இ	i	ச்	c
ஈ	ī	ஞ்	ñ
உ	u	ட்	ṭ
ஊ	ū	ண்	ṇ
எ	e	த்	t
ஏ	ē	ந்	n
ஐ	ai	ப்	p
ஒ	o	ம்	m
ஓ	ō	ய்	y
ஔ	au	ர்	r
		ல்	l
		வ்	v
		ழ்	ḻ
		ள்	ḷ
		ற்	ṟ
		ன்	ṉ

Note: Composite characters are formed by adding consonants and vowels together. For instance, Tamil letter க is transliterated as *ka* as க = க் (k)+ அ(a) In this way there are 216 composite Tamil letters are formed by composing 18 consonants with 12 vowels, and the composite letters will be transliterated accordingly.

1 INTRODUCTION

This chapter introduces the research study and the thesis structure. The sections of this chapter covers the research problem, research objectives, constraints and challenges, the scope for this research. In addition, this chapter also provides a introduction to the Tamil language and Tamil language computing.

Natural Language Processing (NLP) is a branch of artificial intelligence (AI) that helps computers understand, interpret and process human languages. NLP technologies have become an integral part of human life. Currently, these technologies are even able to automatically generate meaningful text on a given theme (Floridi and Chiriatti, 2020),[1] a task which has been said that only humans could do. The idea of understanding human communication by machine was seeded in 1950 (Turing, 1950). However, it has gained more focus in the last thirty years, especially with the web and social media boom. Besides, the study area of human-centric applications such as personal advisory systems, live speech translation between languages, and behavioural analyses has attracted leading businesses to invest in NLP. Because of all this, NLP has been growing very fast in recent times. It is expected to grow from US$ 3 billion markets in 2017 to over a US$ 43 billion market by 2025.[2]

According to Hirschberg and Manning (2015), the vast increase in computing power, the availability of enormous amounts of data, the development of highly successful machine learning (ML) methods and a much richer understanding of the structure of human language and its deployment in social contexts are the key factors that have led to the recent advancement in the field of NLP.

Apart from a handful of languages such as English, German, Arabic, and Chinese, which are either spoken by a large population or by a population with high purchasing power, less advancement has been observed in other languages, including South Asian languages such as Tamil. Not many linguistic and language resources and NLP tools with reportable results have been developed for these languages or are available in the public domain. Having said that, in very recent times, Tamil has been getting traction, and giant corporates such as Amazon have been developing Tamil interfaces and promoting their services among the Tamil population.

[1] https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3

[2] https://www.statista.com/statistics/607891

Away from big businesses concerns and interests, academia has been pushing research studies on low-resource languages and inventing technologies and techniques to create resources and tools for these sorts of languages. Tamil is one such low-resource language as many vital resources for Tamil language processing and applications such as translators, sentiment analysers, and even good spell and grammar checkers are still lacking. Further, even the little advancements in Tamil language technologies are not available for public use in most cases. This is the context in which I have endeavoured to develop a deep syntactic parser for the Tamil language.

Syntactic parsers find syntactic relations between the words in a sentence, then map these to a tree, a graph, or other structure. This syntactic analysis is useful for various NLP applications such as natural language dialogue systems, grammar error checkers, sentiment analysers, machine translators *etc.*. For instance, Armengol-Estapé and Costa-jussà (2021) show how linguistic information — syntax and semantic information — have been used to improve neural machine translation in low-resource settings with a morphologically different language pair — English to Nepali. Although there are attempts to learn syntactic and other linguistic features automatically using deep neural networks, the technology and understanding of different deep neural architecture are at the experimental level, and the support of linguists is still required to do such exercises (Linzen and Baroni, 2021).

The level of syntactic information captured varies depending on the approaches and grammar formalisms employed. Syntactic parsers that mainly identify clause boundaries and phrases are called shallow parsers or chunk parsers (Sha and Pereira, 2003; Ariaratnam et al., 2014).[3] Abney (1992) includes within the analysis yielded by shallow syntactic parsers the relations that chunks have with the main verb. Another type of parser yields a deeper level of analysis and captures argumentative, attributive, and coordinative relations between the words of a given sentence. The latter approach is considered a deep syntactic parser (Ballesteros et al., 2016). These deep syntactic parsers provide more syntactic information than just the clause/phrase level arrangements, and this sort of deep information is more useful for the development of NLP applications (Ohta et al., 2006; McCord et al., 2012), and cross-linguistic analyses (Futrell et al., 2015).

1.1 The Problem

1.1.1 The need for a syntactic parser

Integrating linguistic information, specifically syntactic information, into language processing tools and applications improves their accuracy. This has been proven for appli-

[3] `http://ltrc.iiit.ac.in/analyzer/tamil/`

cations such as machine translators (Habash, 2007; Li et al., 2017), as well as natural language understanding (McCord et al., 2012; Ohta et al., 2006). It is also shown that integrating syntactic and semantic information explicitly for the training of pre-trained models such as Bidirectional Encoder Representations from Transformers (BERT) improves the model's performance (Zhou et al., 2020), even though some of the linguistic information will automatically be learned during the model training. This constitutes evidence that data annotated with syntactic information are important for the development of NLP applications. In addition, such resources are useful for linguistic studies and computer-assisted language learning. These resources are crucial for languages such as Tamil, which is morphologically rich and has a highly free word order (Futrell et al., 2015). Apart from off-the-shelf language-agnostic syntactic parsers such as Stanza (Qi et al., 2020), and TranKit (Nguyen et al., 2021), no deep syntactic parser has been developed for Tamil. The available off-the-shelf language-agnostic parsers show comparatively low accuracy for Tamil. This is true even for a recently-developed parser (Grünewald et al., 2021) using the pre-trained language model XLM-R. They report that the low-resourced nature of Tamil hinders parsing accuracy.

1.1.2 Constraints and Challenges

Tamil is considered a low-resourced language in terms of what tools, applications and annotated resources are available (Bhattacharyya et al., 2019). There are few publicly available resources such as treebanks, annotated corpora, or linguistic analysis tools such as POS taggers and morphological analysers.Tamil is a Southern Dravidian language spoken natively by more than 78 million people across the world,[4] including in India, Sri Lanka, Malaysia, Singapore, South Africa, Mauritius, Fiji, and Burma (Krishnamurti, 2003). It is also one of the top 20 languages in the world based on population count (Simons and Fennig, 2017).

Chapters 2 and 3 outline the available resources and tools for POS tagging and morphological analysis. More importantly, there are still no quality benchmark or gold resource sets available. Given this, studies cannot be compared in a meaningful and acceptable way.

A review of the literature on the development of syntactic parsers for Tamil is given in Chapter 4. Based on that, it became clear that there are no computational grammars or deep parsers available for Tamil. However, a shallow parser is available online, which is discussed in more detail in Chapter 3. There are two Universal Dependencies (UD) treebanks with a total of 1,154 annotated sentences that are publicly available, one of which has tokenisation and annotation issues. Apart from these two UD treebanks, there is another treebank consisting of 938 sentences annotated using the AnnCorra

[4] `http://www.languagesgulper.com/eng/Tamil.html`

dependency scheme (Devi, 2016) available for research purposes. This only marks inter-phrasal relationships and no intra-phrasal dependencies; in other words, functional dependencies are not marked.

Currently, most research studies are being carried out in isolated or sparse and small research groups, and the resulting products are often in different formats and standards. For instance, more than 20 different tag-sets are being used for Tamil by different research groups. Another issue concerning NLP tools is that they are being developed according to their own separate standards. Such situations are not beneficial for under-resourced languages, as such languages need to have or demonstrate interoperability among resources and tools so that the few resources and tools that are available can be compiled and used efficiently.

Moreover, factors such as lack of funding, human resources, appropriate technology, and, more importantly, the lack of academic recognition by the scientific community have contributed further to the low-resourced status of Tamil.

1.2 Research objectives

In light of the clear gap as briefly reviewed above, I decided to develop a deep syntactic parser for the Tamil language to support the development of NLP applications and linguistic analyses. While the parser needs to render a reportable level of accuracy, most importantly, results have to be well interpretable. On the other hand, NLP application development that is machine learning-based or deep learning-based require a large amount of annotated data. Unsupervised approaches to parsing and the interpretability of deep learning approaches are still being researched. It was decided to develop the following two types of parsers considering the needs and the constraints outlined in Section 1.1.2:

1. A **Data-driven parser**: to support machine learning-based or deep learning-based NLP application development. This requires a significant amount of annotated data, especially in parsing to cover different syntactic constructions. Approaches such as transfer learning and multilingual learning can be used to overcome this data scarcity problem to some extent.

2. A **Grammar-based parser**: to support deep linguistic analyses, and to understand and interpret when and what things go wrong. This parser is not only useful for linguistic analysis but also for application development such as small scope and limited vocabulary machine translation, text generation and computer-assisted language learning. I have used the LFG grammar formalism and the Xerox Linguistic Environment (XLE) toolkit to develop the grammar-based parser.

In order to develop these two parsers, I also developed and created the following

set of tools and resources for Tamil, which in turn also necessitated the development of several pre-processing and post-processing tools.

- **Part of Speech tagger for Tamil**: this is required to annotate a given dataset with their grammatical categories. This is discussed in Chapter 2.
- **Morphological analyser**: Tamil is a morphologically rich language, and many syntactic clues are also included in its morphology. Therefore, these morpheme information have to be elicited. Thus, a morphological analyser using a Finite-State approach has been developed for both a grammar-based parser and data-driven parsing. The development of a morphological analyser and the relevant challenges it posed are discussed in detail in Chapter 3.
- **Treebank for data-driven parsing**: A data-driven parser requires a significant size dataset for training and testing. Since the available data has not been enough, I have developed a dataset using the Universal Dependencies framework (UD) because this has been widely used for deep learning-based applications. In addition, a community of scholars works in this framework and a wide range of processing and development tools that work with UD. The development of the Tamil treebank is discussed in detail in Chapter 6.

1.2.1 Scope

Tamil has been evolving for a long time and has undergone grammatical change over the millennia. Specific constructions that existed in the past are not used in the present context. Modern Tamil is known for its diglossic nature. The spoken and written varieties referred to as the low vs high varieties differ phonologically and morphologically from each other. The spoken varieties further vary among different regions. The style of writing also varies based on genres such as stories, news, letters, *etc.*. This range of variation adds more complexity to the syntactic analysis and requires solid linguistic expertise in order to be analysed.

In light of this, it was decided to narrow the scope and focus on contemporary texts published after the year 2000 and on formal written Tamil texts, which mainly included news and hardware operational manuals.

I have created a rule-based parser based on data from ParGram (this is covered in Chapter 5), which is also used to create grammars for dozens of other languages in order to achieve deep cross-lingual analysis. To maintain the parallelism, I translated existing ParGram sentences into Tamil. Apart from these sentences, I have also made use of sentences from a Tamil textbook to ensure that natural constructions in Tamil are also captured. In this way, my grammar-based parser can parse both structures captured in ParGram and Tamil textbook sentences.

I have compiled a news dataset of 1,000 sentences and annotated it using the Universal Dependencies to train the data-driven parser. However, because a set of 1,000 sentences is not enough to train a neural network, I have also tried an approach called multilingual learning to improve the parsing accuracy. These are discussed in Chapter 6. Since the data-driven parser is built on top of a character-based embedding, the parser will not suffer much without the presence of vocabulary and can provide an analysis for any given sentence, even if, at times, these analyses may be wrong.

1.2.2 Deepness of my parsers

Just as other deep syntactic parsers in literature, the depth that determines my parser is the amount of morphosyntactic information extracted from a given sentence. It is essential to mention that this interpretation of depth differs from what Chomsky (2015) referred to as the deep structure in syntax.

The grammar-based parser provides information as defined by the ParGram annotation scheme. I have extended and added a few new features to the ParGram annotation scheme in order to capture the morphosyntactic information of Tamil, as discussed in Chapter 5. The parser extracts information such as argumentation structure, part of speech (POS), morphology, tense, aspect, mood, person, number, gender, definiteness, and constituency structure. In addition, it also captures constraints and well-formedness checks over the word forms.

The data-driven parser elicits information according to the Universal Dependencies annotation scheme, as reviewed in more detail in Chapter 6. These details include reference to argument structure, POS, morphology, and lemma. In addition to the standard features that form part of the UD guidelines, I have also added language-specific features.

1.3 The Tamil language

Tamil is a Southern Dravidian language spoken natively by more than 78 million people across the world,[5] including in India, Sri Lanka, Malaysia, Singapore, South Africa, Mauritius, Fiji, and Burma (Krishnamurti, 2003). It is also one of the top 20 languages in the world based on population count (Simons and Fennig, 2017). Tamil has more than 2000 years of continuous and unbroken literary tradition (Hart, 2000) and has been recognised as a classical language by the Government of India. It has the longest literary tradition among all the Dravidian languages (Lehmann, 1998). It is an official language of Sri Lanka and Singapore and has official regional status in the states of Tamil Nadu and Pondicherry in India. It has also been recognised as a minority and

[5] `http://www.languagesgulper.com/eng/Tamil.html`

indigenous language in several countries, including Malaysia, Mauritius, and South Africa. Tamil is taught in schools in several countries, including Australia, Canada, and the United Kingdom.

The earliest classical Tamil is called *sangattamiḻ*. Modern Tamil is known for its diglossic nature whereby the spoken varieties, referred to as the low variety or *koḍuntamiḻ* and the written variety, also referred to as the high variety or *centamiḻ* (Britto, 1991; Krishnamurti, 2003), differ phonologically and morphologically. The spoken forms of Tamil vary depending on the regions where they are spoken. This is mainly due to language contact and politics (Schiffman, 2008). Within Sri Lanka, there are several spoken varieties of Tamil. At times, one encounters cases where one speaker may not understand the other. Further, no common agreement has been made among different governments (at least between Sri Lanka and Tamil Nadu in India) on the choice of Tamil words for certain technical terms. Consequently, terminological variation is also present. Although there are variations in spoken Tamil, the linguistic structure of written Tamil remains mostly uniform across the different regions.

1.3.1 Tamil Alphabet

The Tamil language has its own script, which is nowadays referred to as Tamil script and follows the Abugida or Alphasyllabary writing system where a pure consonant and a vowel are written together as one unit or syllable. For instance, the consonant த் (*t*) together with the vowel உ (*u*) form the single unit, a composite character து (*tu*). The alphabet has 12 vowels, 18 consonants (non-composite characters), and 216 composite characters, which are formed when combining pure consonants and vowels together as a single unit. Tamil also has a special character ஃ (*ah*) — a guttural, which is categorised neither as a vowel nor as a consonant but is in the alphabet. The University of Madras lexicon describes this letter as "The 13$^{\text{th}}$ letter of the Tamil alphabet occurring only after a short initial letter and before a hard consonant". In addition to this total of 247 Tamil letters, some letters from the *Grantha* alphabet, which is the script used to write Sanskrit in South India, are also widely used along with the Tamil alphabet.

Apart from the alphabet, Tamil has its own native numerals, which include whole numbers and fractions, metrics, and symbols for various concepts such as time that are still being documented. It is, however, unfortunate that most current Tamil speakers are not aware of these symbols, as they are not included in the learning curriculum at schools.

1.3.2 Tamil Grammars

The grammars of Tamil may be divided into ones that are composed by native scholars and those that Europeans have written to facilitate the acquisition of the Tamil lan-

guage, mainly by foreigners (Pope, 1979). *Tolkāppiyam* is identified as the very first scholarly work on Tamil grammar by a native scholar (Hart, 2000). The date of this publication is not exact, yet it is believed to have been published more than 2500 years ago. This is considered to be a derived work of an even older work called *Agastyam* (Pope, 1979), the whole work of *Agastyam* is not extant. The other notable and still widely used and quoted Tamil grammar is called *Nannūl*, which is dated back to the 13th century. Modern-day Tamil grammar textbooks used in Sri Lanka are themselves based on *Nannūl*. Apart from *Tolkāppiyam* and *Nannūl*, several derived works have been published by native scholars from time to time. An important factor about these works usually accommodates the evolution of the Tamil language. These old grammars are written in the form of verses that only Tamil Pundits can interpret. There are nonetheless well-known commentaries that discuss these books available for common people. Dictionaries, treatises on medicine and literature have also been written in the form of verses until the 16th century. From the 16th century onwards, Europeans and others, mostly missionary scholars, started publishing books in Tamil and writing Tamil grammar books. However, since the 19th century, no notable complete linguistic studies have been done on Tamil (Annamalai et al., 2014). On the other hand, the language has evolved significantly due to modern subject areas, new kinds of literature, technological advancements, and influences from other languages, particularly as a result of global communication and movement. *e̱luttu* (orthography), *col* (morphology+syntax), *pāeruḷ* (semantics), *yāppu* (prosody), *aṇi* (rhetorical embellishment) constitute the five key parts of Tamil grammar (Shanmugadas, 1982; Nachinarkkiniyar, 1937; Senavaraiyar, 1938). Not all of the grammars cover all of these parts. For instance, the widely used *Nannūl* covers only the first two.

Tamil is an agglutinative language whose grammatical parts are suffixed to root words. These suffixes can be identified and separated from the rest of the word to determine the root words. Tamil displays a relatively free constituent order, though it primarily follows a Subject-Object-Verb (SOV) structure in formal writing. Using a corpus-based study, Futrell et al. (2015) show that Tamil has the highest word order freedom when compared to the 100+ languages that are available in the Universal Dependencies Treebank.

When it comes to the word classes or Part of Speech available, modern linguists classify Tamil words into *peyar* (noun), *viṉai* (verb), *peyaraṭai* (adjective), *viṉaiyaṭai* (adverb) and *iṭaiccol* (particle) (Nuhman, 1999; Annamalai et al., 2014). Nominative, accusative, dative, instrumental, sociative, locative, ablative, genitive and vocative are the nine cases that mark nouns as described in the literature (Lehmann, 1993). In addition to the case features, all nouns in Tamil can be categorised into *uyartiṇai* (rational) and *aḫ̱riṇai* (irrational). Entities marked as rational are those perceived as being able to think on their own, while the rest are termed as irrational. For instance,

animals, trees, and furniture items are considered irrational, while adult humans and gods are categorised as rational. This (ir)rationality based marking differs from splits in terms of human vs. non-human, or animacy. For instance, infants are considered irrational just as animals or inanimate objects, even though infants are human and animate. An adult human entity can also still be marked as irrational if they behave in an insane manner. This marking is in turn reflected in the noun-verb agreement.

Tamil verbs have complex morphosyntactic relations, which take auxiliary items such as question particles, emphatic particles, and conjunctive markers, in addition to regular inflections such as tense, person, gender, number, and honorifics to build a surface form. A detailed account of both the nominal morphology and verbal morphology is given in Chapter 3. Core Tamil syntax are then outlined in Chapter 4.

1.3.3 Tamil and other Dravidian languages

As a Dravidian language, Tamil belongs to a family of 26 languages that are natively spoken by more than 220 million people around the world. Among these 26 languages, Tamil, Malayālam, Kannada and Telugu have a long written literature (Krishnamurti, 2003). Asher, in the preface of Suseendirarajah (1999) claims that knowledge of Tamil, specifically Jaffna Tamil, a variety spoken in Sri Lanka, is a vital element that forms part of the understanding of the Dravidian family of languages. This is particularly so since certain features of the language have been preserved in Jaffna Tamil that have been lost elsewhere. Subbārāo (2012) claims that Dravidian languages are the most homogeneous set of languages in the Indian sub-continent in that they display syntactic phenomena which exhibit very little syntactic variation. While this is true of the syntax, the different Dravidian languages, in turn, vary morpho-syntactically in a significant way. For instance, Malayalam does not have subject-verb agreement for person, number and gender features. Telugu has singular human male marked separately from the rest, i.e., human female, animal, place, *etc.*. Tamil, in contrast, displays subject-verb agreement and a wider array of gender values. Therefore, data from one language cannot just be used for the other, even though they belong to one language family. On the other hand, one can think of using tools and techniques that have been developed for other Dravidian languages for Tamil. However, there is unfortunately not much work on other languages either. Moreover, similar to the challenge faced in the case of Tamil, most of the work on other Dravidian languages has also not been made public. Therefore, whatever ends up being produced and developed as part of this research on Tamil should also be useful for other Dravidian languages.

1.3.4 Encoding Tamil letters

Apart from the information about how the Tamil alphabet works, understanding how the graphemes are encoded in computers using Unicode[6] is important in order to develop language processing tools and to write morphological and orthographical rules. Each syllable in Tamil has a single code point. For instance, all the vowels and pure consonants in the Tamil alphabet are each encoded with a single code point. All the composite characters, on the other hand, are encoded with two Unicode points; one to represent the consonant letter, and the other to represent the vowel modifier (also called dependent vowel sign),[7] which is applied to form the composite. For instance, து (*tu*) will have two Unicode points,[8] one for த (*t*) and the other for the ு (*u*)- vowel modifier. Similarly, தொ (tou) also has two Unicode points corresponding to த (*t*) and ொ (*ou*) even though it is represented by three glyphs. Typing these vowel modifiers and handling them as part of computer programming is not a straightforward task.

The Tamil script was first encoded in the Unicode version 1.1 in the year 1993.[9] Since then, several Tamil language processing tools have been built around the world. Prior to the introduction of Unicode, ASCII encoding was used along with special fonts to render Tamil graphemes. Although this is not useful for Tamil language processing, this obsolete approach is sometimes used to create text. This adds an extra level of challenges when processing Tamil texts since these differences are not visible to the naked eye.

Even though Unicode encoding was introduced in 1993, up until 2008, most of the studies were about enabling Unicode in applications and sorting out issues in encoding *etc.*, except for a few works such as Optical Character Recognition, the creation of dictionaries, and so on. Since then, one has been able to find some work on Tamil computing in the direction of the development of linguistic tools such as Part of Speech development, Chunker, *etc.*. Most of these research studies have been published in Tamil Internet Conferences organised by the International Forum for Tamil Information Technology (INFITT), which was founded in 1998. In what follows, we outline the list of some key institutions which have been conducting research related to Tamil computing.

1.3.5 Institutions working on Tamil language computing

Several institutions around the world conduct research on Tamil Natural Language Processing, including universities and research centres, government-funded initiatives and self-motivated open source initiatives.

[6] https://home.unicode.org/

[7] `https://unicode.org/charts/PDF/U0B80.pdf`

[8] https://unicode.org/charts/PDF/U0B80.pdf

[9] `https://www.unicode.org/standard/supported.html`

Anna University – K B Chandrasekhar Research Centre (AU-KBC),[10] and Amrita University[11] are key research institutions that have contributed to the area of Tamil NLP, particularly the area of text processing. Other NLP research, including Text To Speech, and Optical Character Recognition, is being done in other institutions such as the SSN College[12][13] and the Indian Institute of Science in Bangalore.[14] State-owned institutions such as the Centre for Development of Advanced Computing (CDAC)[15] conduct many research studies related to Tamil NLP and other South Asian Languages. Apart from research-based institutions, state institutions such as the Department of Technology Development for Indian Languages (TDIL)[16] and the Tamil Virtual Academy[17] have through the years facilitated the research in these areas and have aggregated available tools and resources.

In Sri Lanka, the University of Moratuwa,[18] the Language Technology Research Laboratory (LTRL)[19] of the University of Colombo School of Computing, and the University of Jaffna[20] all conduct research studies related to the Tamil NLP.

Apart from India and Sri Lanka, universities in other parts of the world, such as the Charles University in Prague[21] also conduct research on Tamil NLP. Community-led initiatives such as Thamizha![22] and the Free Tamil Computing forum[23] also contribute to Tamil NLP via the development of small language processing tools.

I have communicated and collaborated with personnel from most of the institutions mentioned above to better enhance and increase the synergy that exists. I have also co-organised a three-day workshop on Tamil Treebank annotations with key personnel in these institutions and invited key scholars from abroad. All these collaborations have helped me to mobilise tools and resources as and when available and required.

1.4 Thesis outline

This introductory chapter outlined the need for a Tamil syntactic parser, and how such parsers have developed. In addition, it gave a brief introduction to the Tamil language and the state of affairs in the Tamil computing world.

[10] www.au-kbc.org
[11] www.amrita.edu/center/computational-engineering-and-networking
[12] www.ssn.edu.in/Speech_Lab/
[13] speech.ssn.edu.in/
[14] mile.ee.iisc.ac.in/projects.html
[15] www.cdac.in/index.aspx?id=mlingual
[16] www.tdil.meity.gov.in/Research_Effort.aspx
[17] www.tamilvu.org/en/content/research-tools
[18] `https://uom.lk/`
[19] `http://ltrl.ucsc.lk/ta/`
[20] `http://www.csc.jfn.ac.lk/`
[21] `https://ufal.mff.cuni.cz/search/node/Tamil`
[22] thamizha.org
[23] groups.google.com/forum/#!forum/freetamilcomputing

The rest of the chapters in this thesis are self-contained, where the required background, literature, methodology, contributions and future work are included in each of them separately. An outline of Chapters 2-7 is given below. Among these chapters, Chapters 2-6 are the key chapters that describe the development of support tools and parsers. Figure 1.1 shows how these key chapters (or tools) are connected. As shown in the picture, the output obtained in Chapter 2 is used to develop the data-driven parser (outlined in Chapter 6), and the output of Chapter 3 is used to develop both the grammar-driven parser (outlined in Chapter 5) and the data-driven parser (outlined in Chapter 6). Chapter 4 gives the foundation for both Chapter 5 and Chapter 6.

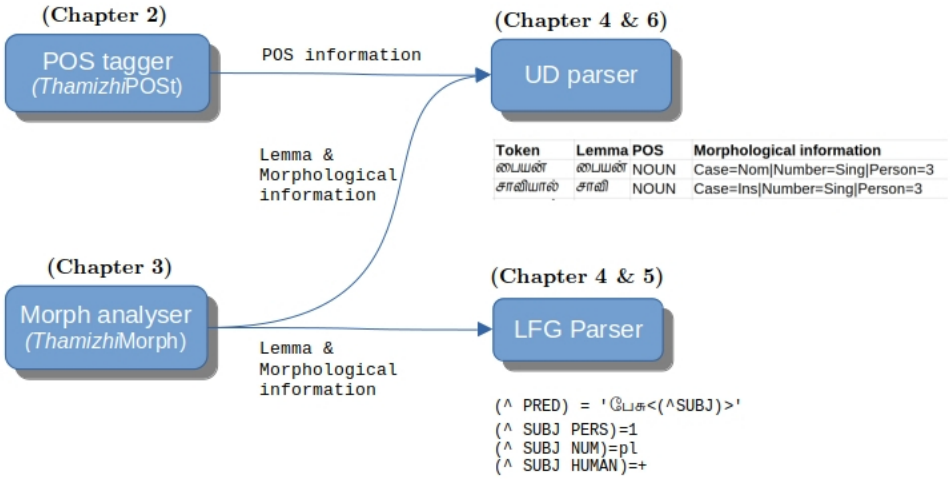

Figure 1.1: Connection between the key chapters (Chapters 2-6) of this thesis.

- Chapter 2 constitutes the development of a Part of Speech (POS) tagger and related work. I use the POS tagger to annotate tokens in the treebank with POS annotations before passing them to the data-driven dependency parser, as POS information is essential to determine the dependency annotation.

- Chapter 3 describes how I developed a morphological analyser cum generator. This chapter thus briefly discusses the morphology of nouns, verbs, and other particles in Tamil to provide the relevant foundation. It then proceeds with a discussion of the challenges encountered in developing a morphological analyser for Tamil.

- Chapter 4 covers the required concepts and background related to treebanks and syntactic parsing, in addition to the relevant literature. This chapter also gives an overview of Tamil syntax that is required to develop a grammar-driven parser and a data-driven parser that are outlined in Chapter 5 and Chapter 6, respectively.

- Chapter 5 discusses how a grammar-driven parser is developed using the Lexical-Functional Grammar formalism and ParGram specification.
- Chapter 6 chapter outlines the development of a data-driven parser using the Universal Dependencies formalism and deep learning approaches.
- Chapter 7 concludes with an outline and summary of contributions.

2 PART OF SPEECH TAGGING

Identifying words' lexical category or Part of Speech (POS) is critical for many natural language processing systems, including syntactic analysis (Jurafsky and Martin, 2017; Petrov et al., 2012). We need to know the POS of words in a given sentence to start the syntactic analysis, irrespective of the formalism we use. Existing Tamil POS taggers are either inaccurate or use a different POS tagset than the Universal Part of Speech tags Petrov et al. (2012) used in the Universal Dependencies. Therefore, we developed a POS tagger that tags words with their lexical category. A different set of labels (POS tagsets) are being used based on the required depth of analyses. This chapter will discuss the current Tamil POS tagging efforts and how I developed a POS tagger for Tamil.

Lemmatisation is the process of identifying the word's lemma or lexical root. This process is also a crucial application for syntactic analysis. For instance, it is essential to know the root of a given verb to identify its valency, and it can be used to check the correctness of the given syntactical analysis. This chapter also describes how I trained a lemmatiser to identify the lemma of a given word.

2.1 Literature survey

This section provides a detailed analysis of existing efforts in developing POS tagsets, POS taggers, and POS datasets for Tamil. This analysis also compares various approaches that have been used to develop taggers for the Tamil language.

2.1.1 POS Tagsets

A Part of Speech Tagset is a collection of labels used to annotate the lexical category of a token. Researchers have no common consensus on how to identify a tagset for a language. Several different tagsets have been proposed, which differ in granularity, naming convention, and the purposes for which the tagset is used. For instance, Petrov et al. (2012) have tabled tagsets used in 25 different treebanks. The number of POS tags in a tagset ranges from 294 POS in the Sinica treebank (Chinese) to 11 in the SynTagRus Treebank (Russian). Fine granularity tagsets can provide more lexical

Table 2.1: Tagsets used for POS tagging of Tamil

Citation	**tagset name**	**no. of tags**
Dhanalakshmi et al. (2009a)	Amrita	32
Pattabhi et al. (2007)	BIS	High-level = 11 and Low-level = 42
Sarveswaran and Mahesan (2014)	New tagset	High-level = 7 and Low-level 10
Nivre et al. (2016)	UPOS	17

information. On the other hand, depending on the availability of a higher order tool, such as a morphological analyser or chunker, we may not need to go for too many tags, especially in the case of morphologically complex languages, where a significant amount of information can be extracted with the use of a morphological analyser. As discussed by Sarveswaran and Mahesan (2014), tagsets can be flat or hierarchical. The most appropriate tagset can be chosen depending on the application for which it is being made. It is always possible to map from a hierarchical tagset to a flat tagset; however, the opposite is not always possible. Table 2.1 shows the different tagsets developed for Tamil so far.

2.1.2 POS Taggers

A POS tagger is a basic language processing tool that identifies the part of speech of each word in a given text. Several attempts have been made to develop POS taggers for Tamil and other Indian languages. Harish and Rangan (2020); Kumar and Josan (2010); Antony (2011) have surveyed the POS tagger development for Hindi, Punjabi, Malayalam, Bengali, Telugu, Tamil and Kannada. Among these, Malayalam, Telugu, Kannada and Tamil are Dravidian languages.

Various approaches are being followed in the development of POS taggers. The survey papers have outlined the following approaches used for Indian languages, Decision Tree (CN2), Maximum Entropy Markov Model, Conditional Random Field (CRF), Hidden Markov Model (HMM), Support Vector Machine (SVM), and neural network-based approaches such as RNN, LSTM and biLSTM. The accuracy of these approaches ranges from 64.91% to 95.63%. Neural-based approaches, SVM and HMM, have been giving relatively good accuracy compared to other approaches.

At least seven research studies have reported Tamil POS tagger development. Among these, the following attempts listed below are reported in Rajendran (2006) and Antony (2011), yet, the actual paper, report, or further details about these systems could not be obtained:

- TagTamil: This is developed by Vasu Ranganathan using a lexical, phonological approach;
- Ganesan's POS tagger: Ganesan has developed a POS tagger using a dictionary and a morphological analyser based upon the Central Institute of Indian Languages (CIIL) corpus;
- Kathambam POS tagger: This is developed using heuristic rules and Tamil linguistic features.

Apart from these three attempts, the other attempts are reported in Table 2.2, where I have summarised the POS tagging accuracy reported in the respective papers. I could not test them with a common dataset as they use different POS tagsets, and most of the POS taggers are not available for public use. Among the reported work for Tamil in Table 2.2, in Selvam and Natarajan (2009) and Mokanarangan et al. (2016) tools, datasets, or/and any documentation are not available online. Dhanalakshmi et al. (2009a) have published the POS tagged data they used to train their tool, yet, they have not published their tool. As a result, these tools could not be tested with the Universal Dependencies Treebank test dataset[1] which I used to test other tools. Rather a discussion could only be made on the results authors mentioned in their research publication. Avinesh and Karthik (2007) has released all the data and tools online.[2] A recent work by Thavareesan and Mahesan (2020) compares different word embeddings to develop a POS tagger for Tamil. Comparatively, Thavareesan and Mahesan (2020) have reported the best result. The authors have found that Bag of Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF) and fastText work better when compared to Word2Vev and GloVE for Tamil POS tagging.

Stanza (Qi et al., 2020), and TranKit (Nguyen et al., 2021) are not just POS taggers. Instead, they are NLP toolkits, which also have several other features. These tools, however, use very recent technologies and approaches and are continuously being developed. Further, these two are trained using the annotated dataset in the Universal Dependencies repository. The TranKit discussed in Nguyen et al. (2021) is uses a state-of-the-art approach called cross-lingual language modelling with BERT called XLM-R (Conneau et al., 2020). Stanza Qi et al. (2020) is the successor of CoreNLP (Manning et al., 2014) toolkit that supports for various core NLP tasks, including POS tagging of around 100 languages which are available in the Universal Dependencies repository.

[1] `https://github.com/UniversalDependencies/UD_Tamil-TTB/blob/master/ta_ttb-ud-test.conllu`

[2] `github.com/avineshpvs/indic_tagger`

Table 2.2: Scores of POS taggers for Tamil POS tagging

Citation or Name	**Method**	**Accuracy**
Selvam and Natarajan (2009)	Rule-based + Projection and Injection technique	92.56%
Dhanalakshmi et al. (2009a)	Support Vector Machine	95.63%
Mokanarangan et al. (2016)	Graph-based neural approach	87.4%
Avinesh and Karthik (2007)	BiLSTM + Conditional Random Fields	87.0%
Qi et al. (2020)	BiLSTM	82.6%
Thavareesan and Mahesan (2020)	{BoW, TF-IDF, fastText, GloVe, Word2Vec}+SVM and k-NN	96% (99% for seen data)
Nguyen et al. (2021)	Transformers	86.18% and 87.64%

Table 2.3: Available POS tagged datasets for Tamil

Citation or name	**Tagset used**	**Size (no. of tokens)**
Dhanalakshmi et al. (2009a)	Amrita	227,000
AUKBC POS Corpus	BIS	515,283
UD_Tamil-TTB	UPOS	8,856
UD_Tamil-MWTT	UPOS	2,625

2.1.3 Datasets

In this section, I document the list of POS-tagged datasets that have been made available for developing Tamil POS taggers. Unlike in the case of a rule-based POS tagger development, datasets are essential for machine learning-based POS tagger development.

Table 2.3 shows the list of POS-tagged datasets available for Tamil. Among these, Dhanalakshmi et al. (2009a) and UD_Tamil-MWTT[3] are human verified datasets. Though the initial version of UD_Tamil-TTB[4] was a human-annotated set, it was later modified using scripts to accommodate multi-word tagging. Unfortunately, no benchmark datasets, verified and widely accepted datasets for testing, are available for Tamil. Nowadays, however, most systems are being evaluated using test datasets found in the Universal Dependencies repository, despite its shortcomings.

[3]github.com/UniversalDependencies/UD_Tamil-MWTT/tree/master
[4]github.com/UniversalDependencies/UD_Tamil-TTB/tree/master

2.2 *Thamizhi*POStagger

Table 2.2 has shown that there are several POS taggers available for Tamil. However, except for the tools developed in Avinesh and Karthik (2007) and Qi et al. (2020), the rest are not available for reuse. Among these, Qi et al. (2020) is trained with the use of tagged data from UD_Tamil-TTB,[5] which is not human-verified and has tagging issues. In the case of Avinesh and Karthik (2007) it is not clear what the data source is. Avinesh and Karthik (2007) has given only two long sentences using which their system was tested. Further, the accuracy of this tagger is also not that high compared to other Tamil POS taggers.

POS tagging is crucial for syntactic parsing, as all the syntactic analyses are, in one way or another, based on the nature of the POS type. Given the current unsatisfactory standing or performance of tools currently available for Tamil POS tagging, I have trained a POS tagger called *Thamizhi*POStagger, with uses Stanza, to cater to our research needs.

2.2.1 The POS tagset

As outlined in section 2.1.1, several POS tagsets have been used for Tamil. These vary in both their granularity and structure. I have decided to use the UPOS tagset, originally proposed by Petrov et al. (2012), to develop *Thamizhi*POStagger. Below are the two reasons for doing so.

- the number of tags in the tagset is higher in certain tagsets since those try to capture considerable morphosyntactic information in the POS tagset. For instance, in the BIS tagset (Pattabhi et al., 2007), most of the second level of information encodes morphosyntactic characteristics. However, if a reliable morphological analyser is available, this can be used instead of complicating the POS tagset. In our case, we have a morphological analyser (Sarveswaran et al., 2019) and can use this in conjunction with a POS tagger.
- UPOS tagset is derived from a crosslinguistic analysis of several languages and used to annotate more than 100 languages in the Universal Dependencies repository. Therefore, if we tag something using UPOS, we would enable crosslinguistic or at least better comparable analyses while also allowing us to use all the tools developed around the Universal Dependencies repository.

UPOS has 17 POS tags as outlined in Table 2.4, as per Zeman et al. (2020). Among these tags, ADP, AUX, CCONJ, DET, NUM, PART, PRON, SCONJ, PUNCT, and SYM are specified as tags for closed class words, which means we should be able to

[5] https://github.com/UniversalDependencies/UD_Tamil-TTB/tree/master

Table 2.4: UPOS tagset

Tag name	Description
ADJ	Adjective
ADP	Adposition
ADV	Adverb
AUX	Auxiliary
CCONJ	Coordinating conjunction
DET	Determiner
INTJ	Interjection
NOUN	Noun
NUM	Numeral
PART	Particle
PRON	Pronoun
PROPN	Proper noun
PUNCT	Punctuation
SCONJ	Subordinating conjunction
SYM	Symbol
VERB	Verb
X	Other

make a finite list of words for each of these categories. However, in Tamil, we cannot do this because of the following two reasons:

- Conjunctions in Tamil are not always marked categorically; in some cases, conjunctions are resolved morphologically. For instance, உம் in கண்ணனும் (kannanum) 'Kannan.NOM.CONJ' marks conjunction within the word-form, which cannot be categorically marked. The conjunction in this instance becomes subsumed under the tag NOUN; this is determined by the category of the semantic head of the given word.

- ADP represents almost all post-positions. Handling them in Tamil is also challenging, as, most of the time, these are suffixed to the main word. It is in effect for this reason that the taggers developed in Nguyen et al. (2021) and Qi et al. (2020) use a special type of tokenisation called multi-word expansion, with which post-positions are segmented from the main word. In the tagger I have developed, I have handled them without the need to break them. I tagged the word according to the category of the semantic head. For instance, அதுமேல் (atumēl) 'on top of it is marked as PRON.

2.2.2 The approach

As outlined in section 2.1, particularly in Table 2.2, various approaches have been used to develop Tamil POS taggers. These include rule-based, machine learning-based, and statistical-based approaches. I need to implement a POS tagger to aid our syntactic parser development. At the same time, I require the POS tagger to do contextual POS tagging and want to make it as generic as possible.

Rule-based approaches require a large set of rules to capture the context and are not robust enough when it comes to unseen words. On the other hand, machine learning or deep learning approaches require a lot of tagged data to train a POS tagger. Fortunately, we have some 100K POS tagged data for Tamil as outlined in Table 2.3. In addition, there are customisable open source deep-learning frameworks such as Stanza (Qi et al., 2020) available for public use, which can be used to build a POS tagger.[6] Further, the deep-learning approach is the state-of-the-art approach used for various tasks, including natural language processing. Therefore, I decided to use a deep learning approach to develop the POS tagger.

2.3 Methodology

In order to implement the supervised deep-learning approach with which I have developed the *Thamizhi*POStagger, a POS tagger for Tamil, I used an off-the-shelf deep learning framework — Stanza (Qi et al., 2020). This tool is the successor of the widely used CoreNLP (Manning et al., 2014). Stanza is implemented with Recurrent Neural Networks (RNN), specifically consisting of a Bidirectional Long Short-Term Memory (BiLSTM) architecture. Stanza allows us to customise architecture and hyperparameters, including the number of layers, activation function, loss function, learning rate, and many more. It provides a pipeline for parsing, which includes the tools for tokenisation, POS tagging, multi-word expansion, lemmatisation, morphological parsing, dependency parsing and Name-Entity Recognition. I have used all of these tools, except for the Name-Entity Recogniser (NER). Because although NER may be relevant for language processing, I could not find any publicly available datasets to train a NER at the time of developing my parsers. Further, developing a NER was not within my study scope.

In this section, I have outlined the steps that have been followed and the tools used to implement the *Thamizhi*POStagger.

[6] At the time of the development of *Thamizhi*POStagger, trankit (Nguyen et al., 2021) was not yet released. I have hence not explored it much, although it uses very recent and state-of-the-art transformer pre-trained models.

2.3.1 Data Preparation

Stanza consists of a pre-trained model for POS tagging of Tamil. However, this has been trained on the UD_Tamil-TTB treebank, which contains many tagging errors. For this reason, I decided to retrain Stanza with a new set of data verified by humans. There were two such datasets found, namely, the AUKBC POS-tagged corpus[7] and the Amrita POS-tagged corpus (Dhanalakshmi et al., 2009a). Both of these corpora include a large number of POS-tagged tokens to train a deep-learning model. The AUKBC POS-tagged corpus has more tagged data than the Amrita-tagged corpus: 515,283 tokens in total. However, all of its texts are taken from a classical Tamil novel. The names are repeated in the corpus, and there are a notable number of texts of conversations of various characters in the novel. On the other hand, the Amrita POS-tagged corpus contains 227,000 human-verified POS-tagged tokens taken from 19,098 sentences from various contemporary Tamil news texts. I intend to develop a POS tagger for contemporary written Tamil text. Therefore, I made use of the Amrita POS-tagged corpus to retrain Stanza to implement the Tamil POS tagger — *Thamizhi*POStagger.

The next step in the process was to harmonise the Amrita POS tagset and the UPOS tagset to convert the Amrita POS tagged corpus to a UPOS tagged corpus. Since the Amrita tagset consists of 32 tags, while the UPOS tagset consists of 17 tags, as outlined in Table 2.4, the harmonisation was not a straightforward task. Another factor that needs to be considered is that Amrita captures morphosyntactic features at the POS level. For instance, verbs are marked whether they are infinitive or gerundive or adjectival or adverbial etc. in the POS tagging, although these information can be captured using a morphological analyser.

I created a tag harmonisation table to map the Amrita tagset and the UPOS tagset as shown in Table 2.5. The mapping of the common categories such as NOUN, ADV, ADJ, PRONOUN, PROPER NOUN *etc.* was not difficult. However, I could not easily map the Amrita category in one of the UPOS categories for the following categories listed below. I ended up handling these case by case manually. In this way, I eventually converted the Amrita POS-tagged corpus into the UPOS-tagged corpus.

- <QW> - question words were challenging as all the words with a question particle are in the Amrita corpus marked with <QW>, irrespective of whether it is a noun or verb. For instance, வந்தானா (vantāṉā) 'did (he) come' and அவனா (avaṉā) 'is he' are marked with <QW>. Therefore, in the annotated corpus, I looked at each occurrence of <QW> and labelled this manually, whether it was a noun or verb.

[7] `http://au-kbc.org/nlp/corpusrelease.html`

- <RDW> - all reduplicated words, including nouns, verbs, and particles, are marked in the Amrita's corpus: <RDW>. These were also handled separately and marked whether they were nouns, verbs, or particles.

Stanza, as mentioned earlier, is a parsing framework developed around the Universal Dependencies formalism and was used to train the POS tagger. It incorporates a language processing pipeline to which raw text is fed, and then the text undergoes tokenisation cum sentence splitting and multi-word expansion before starting any processing. At this preprocessing stage, data has also been converted to CoNLL-U[8] format. CoNLL-U is the popular format which is used to annotate dependency treebank. This format was first proposed by Buchholz and Marsi (2006) in the name of CoNLL-X (published at the 10th Conference on Computational Natural Language Learning). It was later updated for the Universal Dependencies treebank with a few changes.

Stanza provides a tokeniser and multi-word expander. However, they are trained using the UD_Tamil-TTB treebank. This treebank has tokenisation issues where tokens are inappropriately broken. For instance, அமைக்கப்பட (amaikkappaṭa) 'to be built' has been broken into the following tokens, which is wrong, and does not make any sense: அமைக்க (amaikka) and ப்பட (ppaṭa). Therefore, I trained a tokeniser cum sentence splitter and a multi-word expander using the Amrita corpus.

I developed my own script to convert the POS-tagged data from Amrita POS to UPOS to CONLL-U format. This is the format in which treebanks are stored in the Universal Dependencies repository. A sample of the CoNLL-U text is shown in Table 2.6. Eight different linguistic information are captured, apart from the token_id, in the CoNLL-U format. However, we only need the POS information for the development of the POS tagger. In Stanza, POS and morphology training happen together. However, since I focused only on the POS tagging, I filled all the other fields with dummy data (I will cover the other fields in detail in Chapter 6 when I discuss data-driven parsing). In this way, I converted all 227K tokens with UPOS tags to a CoNLL-U format. Sentence structures were preserved, as it is quite useful to capture the context of a given word.

Table 2.6: CoNLL-U formatted text to train the POS tagger

	Token		**POS**						
1	அவையே	_	NOUN	_	_	0	root	0:root	_
2	ஒருவரின்	_	NOUN	_	_	1	dep	1:dep	_
3	நோக்கமாக	_	ADV	_	_	2	dep	2:dep	_
4	இருக்கக்	_	VERB	_	_	3	dep	3:dep	_
5	கூடாது	_	VAUX	_	_	4	dep	4:dep	_
6	.	_	PUNCT	_	_	5	dep	5:dep	_

[8]`https://universaldependencies.org/format.html`

Table 2.5: The harmonisation of the Amrita and UPOS tagsets

Amrita POS tag	**Description**	**UPOS mapped**
ADJ	Adjective	ADJ
ADV	Adverb	ADV
CNJ	Conjunction	CCONJ
COM	Complementizer	SCONJ
COMM	Comma	PUNCT
CRD	Cardinals	NUM
CVB	Conditional verb	CCONJ
DET	Determiners	DET
DOT	Dot	PUNCT
ECH	Echo words	INTJ
EMP	Emphasis	PART
INT	Intensifier	ADJ
NN	Noun	NOUN
NNC	Compound Noun	NOUN
NNP	Proper Noun	PROPN
NNPC	Compound Proper noun	PROPN
NNQ	Quantity Noun	NUM
ORD	Ordinal	NUM
PPO	Postposition	ADP
PRIN	Interrogative Pronoun	PRON
PRID	Indefinite Pronoun	PRON
PRP	Pronoun	PRON
QM	Question mark	PUNCT
QTF	Quantifier	NUM
QW	Question word	VERB
RDW	Reduplication Word	ADP
VAX	Auxiliery Verb	VAUX
VBG	Gerundive Verbal	NOUN
VF	Finite Verb	VERB
VINT	Infinitival Verb	VERB
VNAJ	Nonfinite Adjective Verb	VERB
VNAV	Nonfinite Adverb Verb	VERB

Table 2.7: Training - Development - Testing sets

Dataset	Number of sentences
Training	13,098
Development	5,000
Testing	1,000

2.3.2 Training

As discussed in Section 2.3.1 19,098 sentences have been converted to CoNLL-U format. These were then divided as a training, development, and testing sets as shown in Table 2.7. These data sets were fed to the Stanza framework to train the tokeniser cum sentence splitter, the multi-word expander, and the POS tagger.

Stanza uses a BiLSTM-based architecture, in which architecture parameters and other hyper-parameters can be customised. We do not need to do much tweaking to the architecture and the hyper-parameters as those are initialised with suitable values that the processing has learned of many languages and with different configurations. In fact, I changed parameters like the learning rate and drop-out rates. However, I did not see any change in the score. Therefore, I used Stanza in its default configuration, except for the parameter 'number of epoch', to train the tokeniser cum sentence splitter, multi-word expander and the POS tagger.

Before the processing stage, all the text needed to be converted to vector forms using an embedding process. There are different ways to embed (Mikolov et al., 2013; Bojanowski et al., 2017), and each of these is good for different tasks. Stanza supports word2vec or character-based word embedding. (Qi et al., 2018) In our case, character-based word embedding was deemed more useful as it can capture morphological features (Bojanowski et al., 2017). Instead of training my own character embedding model, I used a pre-trained character model called fastText (Bojanowski et al., 2017) from Facebook to do the embedding.

2.3.2.1 Tokeniser cum sentence splitter

As the first step, the Amrita corpus was fed to the tokeniser architecture of Stanza in the raw text form as well as in the CoNLL-U format. The neural network will learn to do the tokenisation and identify the sentence boundaries based on how text are structured and sequenced in the Amrita corpus; here, the neural network has been trained to predict whether a given character is the end of a token or the end of a sentence. Once the training is over, the tokeniser and the sentence splitter can be used on raw text to do the tokenisation and sentence splitter.

2.3.2.2 Multi-word expander

The multi-word expander has an encoder-decoder architecture to learn about syntactic units. It also uses a frequency lexicon that has been built during the training processes. However, since we use Stanza, all these processes are abstracted, and we need to feed a corpus in raw and CoNLL-U format. Therefore, in my case, I inputted the Amrita corpus in both of those formats to train a multi-word expander. Further, the default number of epochs for training was thirty. However, when this was increased to a thousand epochs, I found good results.

2.3.2.3 POS tagger

Stanza uses a BiLSTM neural network architecture for POS tagging as well. I inputted the Amrita training and development set to train the POS tagger. The POS tagger also does the morphological analysis or finds morphological features. However, the training data also must consist of them. The training data we used do not have morphological features; see sample data on Table 2.3. Preparing and reviewing morphological features require much labour. Therefore, in my case, I handled morphological features separately using the morphological analyser that I developed; this is outlined in Chapter 3.

2.3.3 Evaluation and Discussion

Table 2.8 shows the accuracy for tokenisation, sentence splitting or sentence identification, multi-word expansion, and the POS tagger - *Thamizhi*POStagger.

Apart from the testing that I did with the test set, I inputted 100 sentences with 2,544 tokens taken from a random website to see how the tokeniser, sentence splitter and POS tagger perform on a random, uncleaned dataset. This data set had some quality issues, which I have not discussed here. Apart from data quality issues, I found the following observations with the tools I trained when it came to an unseen and uncleaned dataset.

- Analysis of the tokeniser:
  - quotation marks are not tokenised: a tokeniser is expected to leave a space between quotation marks and text. This is because the training dataset does not have enough occurrences with quotation marks.
  - யாழ்.நாக (yāl̲.nāka) has not been tokenised. Here the period denotes that யாழ் is a short form of யாழ்ப்பாணம். It may be that I need to add more instances of this particular case to the training data.

- in three instances, the question mark and exclamation mark, were not tokenised. This issue is also due to training data; I need to include more sentences with the question and exclamation marks.

- Analysis of sentence splitter:
  - titles are not properly handled during the sentence splitting. This is because there is no specific end mark at the end of the title clause or sentence to mark the end. Therefore, titles are not broken into sentences. Instead, it is added to the subsequent sentence. Except for this issue, the sentence splitter worked well.

- Analysis of POS tagger:
  - a name acronym சீவி(cīvi) has been tagged as a verb. In fact the lexical meaning of this word is 'make sharp'. Therefore, in a way, the tagger got it correct. However, contextually, this is wrong.
  - proper nouns are identified as nouns when they modify another noun. For instance, *cīn̲a* in சீன வெளிவிகார அமைச்சின் (cīn̲a veḷivikāra amaiccin̲) 'Chinese Ministry of Foreign Affairs' is tagged as noun, while *cīn̲āvukku* in சீனாவுக்கு ஊடாக (cīn̲āvukku ūṭaka) - China.DAT through - 'through China' is marked as proper noun. The reason for this outcome may be that *cīn̲a* is a modifier form *cīn̲ā* 'China' which functions as a noun modifier. Whenever proper nouns function as a noun modifier, they are marked as a noun in the tagged data. I am not quite sure how this has to be annotated, anyway. However, if this way of annotation is correct, which means my tagger has learnt linguistic information from the data.
  - srī in ஸ்ரீ விமல தேரர் (srī vimala tērar) 'his-holiness Vimala Bhikkhu' - a Buddhist monk - is marked as a proper noun, though it is adjective to mark 'his holiness'. The reason for this observation could be that some people have their short name as srī, so it has a function of a proper noun in other cases.
  - interestingly, first name - nisaṅka in the full name நிஸங்க சேனாதிபதி (nisaṅka cēn̲āṭipati) is tagged as a proper noun, correctly, even though lexically it can give the meaning of 'Commander-in-Chief' - a noun.

Table 2.8: The evaluation results

Task	Accuracy
Tokenisation	99.5%
Sentence identification	96%
POS tagging	93.27%

2.4 Conclusion and future work

2.4.1 Conclusion

I have explained how I developed *Thamizhi*POStagger — a POS tagger for Tamil using the neural-based NLP toolkit called Stanza. *Thamizhi*POStagger gives the current best score of 93.27%[9] for Tamil POS tagging using the Universal Part of Speech (UPOS) tagset, among the tools available for public use. I have also outlined the tools and resources available for Tamil POS tagging. As has been shown, very much as is the case with other Tamil NLP tools, not many tools and resources are available for Tamil POS tagging. Even the ones that are available have not been made public. This tool can be used to generate POS-tagged data for developing or aiding other NLP applications such as spell checkers and machine translation engines.

I have made the following contributions with respect to the Tamil POS tagging:

1. I have created a state-of-the-art POS tagger for Tamil to tag data using the popular UPOS tagset. I have made this available as a Python library called thamizhilip (Thamizhi Linguistic Processing Library),[10], which can be easily set up and used in computers.

2. In addition to the POS tagger that can tag using the UPOS tagset, I have also published a POS tagger, which can be used to tag text with the Amrita POS tagset. This integrated feature is useful for doing fine-grained POS tagging. This is also integrated into the thamizhilip python library.

3. I have rendered a harmonisation of the Amrita POS tagset and UPOS tagset

More information associated with the *Thamizhi*POStagger - POS tagger and the annotated data generated can be obtained here: `http://nlp-tools.uom.lk/thamizhi-pos/` and `https://github.com/sarves/thamizhi-pos`

[9] I have also published this result in one of my paper (Sarveswaran and Dias, 2020)

[10] `https://pypi.org/project/thamizhilip` `https://sarves.github.io/thamizhilip`

2.4.2 Future work

The arrival of a new deep learning architecture called Transformer (Vaswani et al., 2017) outperforms most of the existing architectures, including the one we used to develop *Thamizhi*POStagger. One can now explore how this architecture can be used to improve POS tagging. On the other hand, one can argue that the Transformer is too much of an expensive device to use for a POS tagging task. A transformer-based NLP tool kit, namely TranKit, has been published (Nguyen et al., 2021) and is inspired by Stanza. TranKit also has a POS tagging tool and a pre-trained POS tagger for Tamil. I strongly believe that with the data I have prepared, we can achieve better performance for Tamil POS tagging if TranKit is used. Due to time constraints, however, I have not tried this yet. However, it will be worth investing some time to experiment with this in the future.

Further, it will also be worth exploring how much linguistic information is captured through the POS tagger, as briefly done in 2.3.3. One can do this analysis by carefully going through a POS-tagged corpus. However, such an analysis demands strong Tamil language expertise and a linguistic eye.

3 MORPHOLOGICAL ANALYSIS AND GENERATION

3.1 Introduction

This chapter gives an outline of how *Thamizhi*Morph, a morphological analyser and generator for the Tamil language, has been developed using a Finite-State compiler. The chapter also gives an account of the nominal and verbal morphology of Tamil and the challenges faced when handling them in *Thamizhi*Morph. The chapter also documents the previous attempts to develop morphological analysers and generators for the Tamil language. In addition, approach and technology choices are also justified in this chapter. *Thamizhi*Morph has been published online as a usable tool and as a Python library for people to use and extend. This chapter is an extended version of a journal article published as Sarveswaran et al. (2021).

3.2 Tamil Morphology

In this section, I describe the morphology of Tamil in detail. In traditional grammar, Tamil words are categorised into four types, namely nouns, verbs, particles and intensifiers (Senavaraiyar, 1938; Thesikar, 1957). For the purpose of this research to develop a morphological analyser cum generator for Tamil, I have categorised Tamil words into nouns, verbs, adjectives, adverbs and particles. This categorisation renders the development more efficient as these contain the most number of words. Moreover, words in these categories are considered content words, which are used to mark the primary dependencies in the Universal Dependencies formalism. This will also reduce the complexity in drafting rewriting rules or phonological rules to be discussed in later sections. These categories are additionally advantageous as they mostly have non-overlapping morphological rules. These categories will each be discussed in detail in the following sections.

3.2.1 Nominal Morphology

Nouns in Tamil are primarily marked for number and case. These features are expressed as suffixes that are added to a 1. lemma form or 2. oblique form (which will be discussed later in this section). In addition, an oblique form can take euphonic and other phonologically motivated material before the number and case suffix (Caldwell, 1998; Shanmugadas, 1982; Lehmann, 1993).

The number or/and case suffix can be just added to a lemma form, as in example (1) (a) and (b). Apart from such simple suffixation, a glide letter (ய் / வ்) (y/v) may be inserted, as in Example (1) (c), or a consonant (க்/ ப் / ச் / ற்) (k/p/c/r̲) may be inserted, as in Example (1) (d). In addition to these, an assimilation process may also occur where an existing letter is changed to another, as in Example (1) (e), where ல் becomes ற்.

(1) (a) கதிரைகள் (*katiraikaḷ*) 'chairs'

கதிரை	-கள்	
katirai	*-kaḷ*	
chair	PL	

(b) கதிரைக்கு (*katiraikku*) 'to a chair'

கதிரை	-க்கு	
katirai	*-kku*	
chair	DAT	

(c) கதிரையுடன் (*katiraiyudan*) 'with a chair'

கதிரை	-ய்	-உடன்
katirai	*-i*	*-udan*
chair	GLIDE	SOC

(d) பூக்கள் (*pūkkaḷ*) 'flowers'

பூ	-க்	-கள்
pū	*-k*	*-kaḷ*
flower	SANDHI	PL

(e) புற்கள் (*pur̲kaḷ*) 'grasses'

புல்	-கள்	
pul	*-kaḷ*	
grass	PL	

Oblique forms are generated primarily by doubling the last consonant or adding a compulsory oblique suffix. This process makes a stem eligible to receive case markers

in some noun classes (Krishnamurti, 2003). These are referred as *cāriyai* in Tamil grammar books (Nuhman, 1999; Thesikar, 1957). அம் (*am*), அத்து (*attu*), and அற்று (*aṭṭu*) are the widely seen oblique suffixes in texts. Examples for obliques are shown in (3).

Number and case suffixes are bound morphemes. However, in modern Tamil, case suffixes can also be seen separately as free morphemes. Lehmann (1993) also points out the possibility of their occurrence as free morphemes in Tamil.

3.2.1.1 Structure of simple Nouns

The morphology of a simple Tamil noun, without any derivations and compounding, is depicted in the formulas in (2) (a) and (b). There are nouns with two euphonic markings in the corpora used for this research study, as shown in (2) (b). Euphonic morphs such as அன் (*an*) and இன் (*in*) are purely phonological increments (Lehmann, 1993). However, the exact functions of these increments are yet to be understood. (3) shows a few different ways a noun can be formed from an oblique base. A stem can take only one oblique suffix; however, it can take multiple instances of euphonic morphs.

(2) (a) Noun = lemma-form (+plural) (+case)
(b) Noun = oblique-form (+plural) (+euphonic)^2 (+case)

(3) (a) மரங்களினால் (*marangkaḷinaal*) 'by trees'

மரம்	கள்	இன்	ஆல்
maram	*kaḷ*	*in*	*aal*
tree	PL	EUPH	INST

(b) மரத்தினுக்கு (*maratinukku*) 'to a tree'

மரம்	அத்து	இன்	கு
maram	*attu*	*in*	*ku*
tree	OBL	EUPH	DAT

(c) ஆவினுக்கு (*aavinukku*) 'to a cow'

ஆ	இன்	உ	கு
aa	*in*	*u*	*ku*
cow	EUPH	EUPH	DAT

Nouns in Tamil can also be derived from verbs. Although derivational morphology is not within the scope of our work, I have handled some derivations as part of the verb morphology outlined in 3.2.2.

3.2.1.2 Plurals in Tamil

Tamil noun stems are singular by default. The plural suffix is a bound morpheme in Tamil that is marked by கள் (*kaḷ*). However, this marker is also used as an honorary marker in present-day Tamil usage, especially when added to the third person pronoun அவர்-கள் (*avar-kaḷ*) 'he.3SEH'.

3.2.1.3 Cases in Tamil

Traditional grammarians have identified 8 cases including a vocative (Senavaraiyar, 1938; Thesikar, 1957). However, modern linguists (Nuhman, 1999; Paramasivam, 2011; Lehmann, 1993) argue that the instrumental case in traditional grammar should be treated as two, namely instrumental case and sociative case. In the analyser, I adapted this modern classification, and the 9 cases along with the respective markers are shown in Table 3.1.[1]

Several of the above case markers are free morphemes, specifically locative, sociative, ablative and instrumental markers (Lehmann, 1993). For instance, சாவியால் (*saaviyaal*) 'key.INST' can also be written as சாவி மூலம் (*saavi muulam*). However, these free morphemes are not always used to mark cases. *Thamizhi*Morph currently does not identify these free morphemes as case markers. They are marked with their original lexical category. For instance, மூலம் (*muulam*) 'origin' will be marked as a noun instead of as an instrumental case marker. However, one can build a tool on top of *Thamizhi*Morph to easily identify the case marking functions of these free morphemes based on the POS tag of the adjacent words.

Rationality and gender are essential attributes of nouns that are shown in the analysis, as these are valuable for syntactic and semantic processing. Tamil does not have a definite marker. Definiteness is expressed with demonstrative markers or by marking irrational objects with the accusative case marker (Lehmann, 1993). While the object of a sentence may be marked with the accusative case, this is compulsory only for rational objects (Lehmann, 1993; Nuhman, 1999). Therefore, when an irrational noun has an accusative marker, this essentially marks that noun as definite.[2]

[1] Thesikar (1957) lists 28 case markers for the locative case and 10 case markers for the vocative case. Collectively, most of these markers are rarely used in present-day Tamil, and I have, therefore, not included all of them in the current version of *Thamizhi*Morph.

[2] Tamil is a Differential Object Marking (DOM) language. DOM will be covered in detail in Chapter 4.

Table 3.1: Case markers considered for *Thamizhi*Morph

Morpheme	**Morphs / suffixes**	**Example**
Nominative	-	மரம் (*maram*) 'a tree'
Accusative	ஐ (*ai*)	மரத்தை (*marattai*) 'the tree'
Instrumental	ஆல் (*aal*)	மரத்தால் (*marattaal*) 'using a tree'
Sociative	ஒடு, உடன் (*oṭu, uṭan*)	மரத்துடன் (*marattuṭan*) 'with a tree'
Dative	கு,க்கு,அக்கு,உக்கு (*ku,kku,akku,ukku*)	மரத்துக்கு (*marattukku*) 'to a tree'
Ablative	இல்/இன்+இலிருந்து (*il/in+iliruntu*)	மரத்திலிருந்து (*marattiliruntu*) 'from a tree'
Genitive	அது, உடைய, இன் (*atu, uṭaiya, in*)	மரத்தின் (*marattin*) 'of a tree'
Locative	இல், இடம் (*il, iṭam*)	மரத்தில் (*marattil*) 'on a tree'
Vocative	ஆ, ஏ, ஈ (*aa,ee,ii*)	மரமே (*marame*) 'oh a tree!'

3.2.1.4 Nominal paradigm

Rajendran (2009) has proposed a noun paradigm incorporating 26 classes based on their morphophonological properties. Among these 26 classes, nine classes are used to capture the morphophonological rules pertaining to pronouns. I took all 16 classes for non-pronominals from this paradigm to develop *Thamizhi*Morph. For pronominals, I extended nine classes to 38 classes to make the development less complex. Although many pronouns are subject to the same morphophonological rules, they produce different analyses or lexical strings. Therefore, these were sorted into different classes.

The paradigms I used for nouns that are not pronominal are shown in Table 3.2. I have selected one word to represent each class, as Rajendran (2009) does, and the classes are named on the basis of that representative lexical item. Class distinctions are here determined by the last vowel modifier (or vowel) [class 1-3,6-8], consonant [classes 4-5,9-16], or whether it is a two-letter word [class 10 and 11], whether it is a two-letter word and the first letter ends with a long vowel modifier (or vowel)[classes 13 & 14, 6 & 7]. I have also developed a script to classify nouns into their respective classes.[3] Classes 6 & 7, and 13 & 14 have been separated into different classes even though they have the same last character in the orthography. This distinction has been motivated by the fact that nouns in these classes differ in conjugation patterns. Certain nouns can

[3] `https://github.com/sarves/Tamil-Noun-Classifier`

have several conjugational forms, which are widely accepted to mark plurality or case. Tamil grammar texts also accept such multiple forms, and refer it as போலி (pōli) (Nuhman, 1999; Thesikar, 1957). For instance, *naal* 'day' shows conjugations of class 4 and class 14 in present-day Tamil. Therefore, I have included *naal* in both classes.

3.2.1.5 Nominal conjugational forms

I have made use of 36 declension forms for Tamil nouns, which cover plural and case conjugations, along with external *Sandhi* markers. Each noun root takes case markers both in its singular and plural forms. Furthermore, nouns in their dative or accusative forms can also take one of four external *Sandhi* markers. Altogether, this yields 36 nominal conjugational forms. Since it is also common to suffix postpositions to nouns, I am in the process of also including such constructions as part of *Thamizhi*Morph.

3.2.2 Verbal Morphology

The structure of a simple verb in Tamil is shown in Formula (4). The euphonic particle அன் (*an*) is optional. The medial particle is used to realise tense (past, present and future), or to negate the verb (Pope, 1979; Lehmann, 1993; Paramasivam, 2011). The terminal suffix of a finite verb is used to realise multiple types of information such as number, person, gender, and rationality (or status) (Pope, 1979; Lehmann, 1993). However, this terminal suffix cannot be chunked or divided to extract these information. For instance, in (5), *-aan* denotes that the terminal-suffix is third person, singular, masculine, and rational. As for other morphosyntactic features, Tamil has singular and plural values for number, first/second/third person values, and three gender values: masculine, feminine and neuter. In addition to these three genders, a fourth class called 'epicene' is used to mark the third person plural forms of rational entities (Lehmann, 1993). This is what affects the choice of the correct terminal suffix.

(4) Verb=lemma-form+<medial-particle>+(euphonic-particle)+<terminal-suffix>

In addition to simple verbs, Tamil also has complex or compound verbs with more than one verbal root, which may express mood, aspect, negation, interrogative, emphasis, speaker perspective, conditional, and causal relations (Annamalai et al., 2014). Agesthialingom (1971) claims that Tamil can have up to four verbal roots in one verb form. For instance, there are four verbal roots in the complex verb in (5): *vaa* 'come', *koḷ* 'hold', *iru* 'be' and *iru* 'be'. *koḷ* 'hold' and *iru* 'be' in the middle together signal a continuous aspect. In (5), we observe that verbal conjugation is only expressed on the last verbal root of the sequence, where it takes tenses, person, number, and gender (PNG) marking. The preceding verbs appear either in a participial or infinitival form.

Table 3.2: The Tamil nominal paradigm used for *Thamizhi*Morph

No.	**Class name**	**Plural Marker**	**Sample case markers**
1	கடா (*kaṭaa*) 'male goat' or பசு (*pasu*) 'cow'	க்-கள் (*k-kaḷ*)	வை, வால், வுடன், வுக்கு (*vai, vaal, vuṭn, vukku*)
2	எலி (*eli*) 'rat' or நெய் (ney) 'ghee'	கள் (*kaḷ*)	யை, யால், யுடன், யுக்கு (*yai, yaal, yuṭn, yukku*)
3	ஈ (*ii*) 'house fly'	க்கள் (*kkaḷ*)	யை, யால், யுடன், யுக்கு (*yai, yaal, yuṭn, yukku*)
4	நாள் (*naal*) 'day' or கால் (*kaal*) 'leg'	கள் (*kaḷ*)	ை, ா, ு, ோ (*i, aa, u, oo*)
5	பலர் (*palar*) 'many people'	-	ை, ா, ு, ோ (*i, aa, u, oo*)
6	காடு (*kaaṭu*) 'forest'	கள்(*kaḷ*)	ட்டை, ட்டால், ட்டுக்கு, ட்டோடு (*ṭai, ṭaal, ṭukku, ṭooṭu*)
7	வண்டு (*vandu*) 'beetle'	கள் (*kaḷ*)	ை, ா, ு, ோ (*i, aa, uu, oo*)
8	ஆறு (*aaru*) 'river'	கள் (*kaḷ*)	றை, றால், க்கு, றோடு (*r̲ai, r̲aal, kku, r̲ooṭu*)
9	கண் (*kan*) 'eye'	கள் (*kaḷ*)	ணை, ணால், ணுடன், ணுக்கு (*ṇai, ṇaal, ṇuṭan, ṇukku*)
10	பொன் (*pon*) 'gold'	கள் (*kaḷ*)	னை, னால், னுடன், னுக்கு (*nai, naal, nuṭan, nukku*)
11	மாணவன் (*maanavan*) 'student'	ர்கள் (*rḷ*)	னை, னால், னுடன், னுக்கு (*nai, naal, nuṭan, nukku*)
12	புல் *(pul)* 'grass'	ற்கள் (*r̲kaḷ*)	லை, லால், லுடன், லுக்கு (*lai, laal, luṭan, lukku*)
13	முள் (*mul*) 'thorn'	ட்கள் (*ṭkaḷ*)	ளை, ளால், ளுடன், ளுக்கு (*ḷai, ḷaal, ḷuṭan, ḷukku*)
14	நாள் (*naal*) 'day'	ட்கள் (*ṭkaḷ*)	ை, ா, ு, ோ (*i, aa, uu, oo*)
15	மரம் (*maram*) 'tree'	ங்கள் (*ṅkaḷ*)	த்தை, த்தால், த்துக்கு, த்தோடு (*ttai, ttaal, ttukku, ttooṭu*)
16	சுவர் (*suvar*) 'wall'	கள் (*kaḷ*)	ற்றை, ற்றால், ற்றுக்கு, ற்றோடு (*r̲r̲ai, r̲r̲aal, r̲r̲ukku, r̲r̲ooṭu*)

(5) வந்துகொண்டிருந்திருக்கிறான்
vantukoṇṭiruntirukkiraan
vantu-koṇṭiru-iru-kkir-aan
come.VPART-hold_be.VPART-be-PRES-3SMR
'(he) has been coming'

Complex verbs in Tamil can be written as separate tokens, as in (6), or as a single token, as in (7). If a complex verb is written as one token as in (7), then the analyser should provide a proper analysis, including identification of all the verbal roots in it.

In the development of *Thamizhi*Morph I have so far only handled instances of up to two verb roots that are written together and frequently occur in the text. Thus so far, excluding the analysis of verb forms of the type in (5), which include up to four verbal roots. I have analysed complex verbs where the second verbal root functions either as an auxiliary or a light verb. For instance, Sarveswaran and Butt (2019) show how the verb கொடு (*koṭu*) 'give' functions as a light verb when it occurs as the second verb in a complex verb, as shown in (8). I note, however, that more studies are needed to fully analyse the functions and structure of complex verbs in Tamil.

(6) வாங்கச் செய்தான்
vang-a-c sei-t-aan
buy-INF-SANDHI-C do-PAST-3SMR
'(he) made someone buy.'

(7) வாங்கச்செய்தான்
vang-a-c-sei-t-aan
buy-INF-SANDHI-C-do-PAST-3SMR
'(he) made someone buy.'

(8) வாங்கிக்கொடுத்தான்
vāṅkik-koṭu-tt-āṉ
buy-VPART-SANDHI-K-give-PAST-3SMR
'(he) bought for someone'

As a step towards this goal, I have identified a set of verbs which form complex verb forms together with the main verb. I have categorised these according to their structure and function, primarily based on the discussions in Boologarambai (1986), as shown in Table 3.3, and incorporated them into the morphological analyser *Thamizhi*Morph. However, more work is required to identify more complex verbal conjugational forms and incorporate them to *Thamizhi*Morph.

Table 3.3: List of complex verb forms used in *Thamizhi*Morph

Verb types	**Verb roots**[4]	**Structure of a finite verb**
Aspectual verbs	இரு (*iru*) 'be', கொண்டிரு (*koṇṭiru*) 'keep', விடு (*viṭu*) 'leave' முடி (*muṭi*) 'finish'	main-verb+VP +aspectual-verb+TENSE+PNG
Attitude verbs	போ (*po*) 'go' போடு (*poṭu*) 'put', தள்ளு (*taḷḷu*) (push), தீர் (*tiir*) 'solve', தொலை (*tolai*) 'get lost'	main-verb+VP +attitude-verb+TENSE+PNG
Non-attitude or Light Verbs	இடு (*iṭu*) 'put', கொடு (*koṭu)* 'give', பார் (*paar*) 'see', வா (*vaa*) 'come'	main-verb+INF +non-attitude- verb+TENSE+PNG
Modal verbs	வேண்டு (veeṇṭu) 'want', கூடு *(kooṭu*) 'may'	main-verb+INF +modal-verb+TENSE+PNG
Causatives	பண்ணு (*paṇṇu*) 'make', செய் (*sei*) 'make', வை (*vai*) 'cause'	main-verb+INF +causative-verb+TENSE+PNG
Passivisers	படு (*paṭu*) 'suffer', பெறு (*peru*) 'get'	main-verb+INF +passiviser+TENSE+PNG

3.2.2.1 Verbal paradigm

Tamil verbs can be classified on the basis of criteria that can be either morphological, syntactic or semantic (Paramasivam, 2011). Many scholars, including Lisker (1951), Graul (1855), and Arden (1910) have classified verbs on the basis of what their morphophonemic changes they display as part of their conjugations. Graul (1855) has provided an early classification on which other scholars have built their proposals, including Irākavaiyaṅkār (1958) and Sithiraputhiran (2004). Graul's classification has also been adopted for the Tamil lexicon project (Rajaram, 1986). This classification of Tamil verbal lemmas includes 12 categories or classes and is based on the tense markers as displayed on the verbs. I have also adopted this classification and extended it slightly, as shown in Table 3.4; this is because I have further separately incorporated

[4]Note: I have given the literal meaning of the verbs in the column - Verb roots. However, when functioning as an auxiliary or light verb, these meanings may not hold.

complex and irregular verbs. As shown in Table 3.4, when verb forms are conjugated for tense markers, these are not just added to the lemma form via concatenation. Rather changes or new letters may be introduced, and these constructions are handled with the use of alternation rules. For instance, in class 12, when the past tense marker is coined, it is not just added to the lemma, e.g. (நட+த் (*naṭa+t*) 'walk.SANDHI-T. Instead, the letter ந் (*nt*) is inserted as in நட+ந்+த் (*naṭa+n+t*). A similar behaviour presents in other classes as well.

In addition to these classes, I have also identified five irregular verbs that are processed separately within the paradigm: காண் (*kaaṇ*) 'see', வா (*vaa*) 'come', சா (*saa*) 'die', தா (*taa*) 'give', வே (*vee*) 'boil'. These are also shown in Table 3.4.

Table 3.4: The Tamil Verbal Paradigm used for *Thamizhi*Morph

No.	**Class name**	**Past tense marker**	**Present tense marker**	**Future tense marker**
1	செய் (*sei*)	த் (*t*)	கிற், கின்ற் (*kir, kinr*)	வ், உம் (*v, um*)
2	ஆள் (*aaḷ*)	ட் (*ṭ*)	கிற், கின்ற் (*kir, kinr*)	வ், உம் (*v, um*)
3	கொல் (*kol*)	ற் (*ṛ*)	கிற், கின்ற் (*kir, kinr*)	வ், உம் (*v, um*)
4	கடி (*kaṭi*)	த் *(t)*	கிற், கின்ற் (*kir, kinr*)	வ், உம் (*v, um*)
5	அஞ்சு (*angu*)	இன் (*in*)	கிற், கின்ற் (*kir, kinr*)	வ், உம் (*v, um*)
6.1	அடு (aṭu)	ட் (*ṭ*)	கிற், கின்ற் (*kir, kinr*)	வ், உம் (*v, um*)
6.2	நகு (*naku*)	க் *(k)*	கிற், கின்ற் (*kir, kinr*)	வ், உம் (*v, um*)
6.3	உறு (*uru*)	ற் (*ṛ*)	கிற், கின்ற் (*kir, kinr*)	வ், உம் (*v, um*)
7	உண் (*uṇ*)	ட் (*ṭ*)	கிற், கின்ற் (*kir, kinr*)	ப், உம் (*p, um*)
8	தின் (*tin*)	ற் (*ṛ*)	கிற், கின்ற் (*kir,kinr*)	ப், உம் (*p, um*)
9	கொள் (k*oḷ*)	ட் (*ṭ*)	கிற், கின்ற் (*kir,kinr*)	ப், உம் (p,um)
10	நில் (*nil*)	ற் (*ṛ*)	கிற், கின்ற் (*kir,kinr*)	ப், உம் (*p, um*)
11	அபகரி (*abakari*)	த் *(t)*	க்கிற், க்கின்ற் (*kkir, kkinr*)	ப்ப், உம் (*pp, um*)
12	நட (*naṭa*)	த் (*t*)	க்கிற், க்கின்ற் (*kkir, kkinr*)	ப்ப், உம் (*pp, um*)

3.2.2.2 Verbal conjugational forms

Annamalai et al. (2014) have identified 254 forms for each Tamil verb following a rigorous analysis of their corpus of contemporary texts. Some verbs may, however, not take all of the 254 forms. Rajaram (1986) has identified 21 forms for each verb from a pedagogical perspective. On the other hand, Kumar et al. (2010b) claims that a Tamil verb lemma can take up to 8,000 forms if derivations are also considered. In

our *Thamizhi*Morph I have implemented 260 inflectional forms. These forms are the set common to Annamalai et al. (2014) and Rajaram (1986). For each lemma, 260 forms are generated and analysed. However, more forms can easily be added to the system without the need for any additional programming using the Meta-Morph rules (Sarveswaran et al., 2019) discussed in Section 3.6.3.

3.2.3 Adjectival Morphology

In Tamil, there are two sorts of adjectives: pure adjectives and derived adjectives. Pure adjectives are words that are used as adjectives without requiring any suffixation. The derived adjectives have different functions and based on them, they undergo different derivations. I have handled only the attributive type of adjectival derivation, which is formed by adding the suffix ஆன *-aana* 'an adjectiviser' to nouns, as shown in (9), in *Thamizhi*Morph. In addition to the adjectiviser mentioned, there are a few other ones that are also reported in literature (Nuhman, 1999). I will incorporate them in future when I develop a full-fledged derivational morphological analyser.

(9) உயரமான
uyaram-aana malai
tall-be mountain
'tall mountain'

In addition to these two types of adjectival categories, adjectival participles and nouns can also modify a noun, as shown in examples (10) and (11). Even though these do not fall under an inflectional morphological analysis of adjectives, I have handled the derivation of adjectival participles as a part of verbal morphology, given that these are morphologically derived from verbal forms.

(10)

வந்த	பையன்
va-nt-a	*paiyan̲*
come.PAST.ADJ-MARKER	BOY

'The boy who came'

(11)

அமைதிப்	படை
amaiti-p	*paṭai*
peace.SANDHI-P	FORCE

'A peace force'

3.2.4 Morphology of adverbs

Similar to adjectives, adverbs in Tamil come in two types: pure adverbs and derived adverbs. Pure adverbs are word forms that are themselves used as adverbs without

requiring any sort of suffixation; for instance, in (12), the adverb *nēṟṟu* 'yesterday' does not have any suffixation on it. Several adverbs are subsumed under this type, namely, temporal, frequency, place, degree and affirmation (Nuhman, 1999). Manner adverbs fall under the second type, i.e. derived adverbs. These derived adverbs are formed by adding an adverbial suffix *-āka/āy* 'become' to nouns, as shown in (13).

(12)

நேற்று	வந்தான்
nēṟṟu	*vantāṉ*
yesterday	came (he)

'He came yesterday'

(13)

வேகமாக	வந்தான்
veham-āka	*vantāṉ*
speed-ADV	came (he)

'(He) came fast'

In addition to these two types, adjectival participles and nouns can also modify nouns. Even though these do not fall under the remit of inflectional morphology, I have still handled their derivation as a part of verbal morphology.

In this section, I have discussed the morphology of the Tamil language, especially the inflectional morphology of nouns, verbs, adjectives and adverbs. I have also briefly discussed derivational morphology where necessary. It is now evident that Tamil has a complex morphological system. The following sections will outline different approaches used to implement morphological analysers for Tamil, primarily, and how I implemented the proposed Tamil morphological analyser and generator using the Finite-State Morphology. These sections also cover the background required to understand the used technology stack.

3.3 The Finite-State approach

This section aims to give an overview of the Finite-State approach and how it is used to develop morphological analysers and generators. As outlined in Section 3.4, the Finite-State approach has been successfully used to develop morphological analysers for various South Asian languages. Further, I have also used this approach to develop the proposed Tamil morphological analyser. Further, this section will also outline an open-source tool called Foma, which is used to implement the proposed morphological analyser.

3.3.1 Finite-State Morphology

Finite-State machines are widely used in language processing applications, including grammar checking and language validation. The Finite-State Transducer is a type of Finite-State machine which has two tapes instead of one, as is typical in a Finite-State Machine. These are the input tape and the output tape (Jurafsky and Martin, 2017). This idea of two tapes (two-way traversals) has been mapped to morphological analysis and generation. Koskenniemi (1983) came up with an idea of two-level morphology where the one level represents the surface form of the word and the level represents the analysis of a given word, as shown in Figure 3.1. This idea of a two-level morphology is successfully modelled using Finite-State Transducers, where one tape is used for morphological analysis while the second one is used for the generation. For instance, in Figure 3.1, one level shows the surface form 'runs', and the analysis of that word is given in the other level as 'run+V+3p+Sg'. The concept of a two-level morphology and modelling with a Finite-State Transducer (FST) is called Finite-State Morphology (FSM), and it is outlined in Beesley and Karttunen (2003). According to this model, if we input the surface form (for instance, 'runs'), we will get the analysis ('run+V+3p+sp'). The vice versa will also work, and it is called morphological generation.

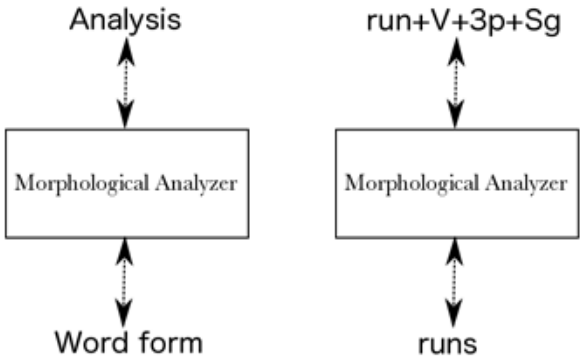

Figure 3.1: Two-level morphology
source: `https://fomafst.github.io/morphtut3.png`

An FSM approach has been widely used to develop successful early applications for morphologically rich languages such as Finnish and Russian (Koskenniemi, 1983; Karttunen and Beesley, 2001). Subsequently, it has been taken up by researchers developing morphological analysers for other languages, including South Asian languages such as Urdu (Bögel et al., 2007), Sindhi (Rahman, 2016), Nepali (Prasain, 2011), and also for the morphologically complex Australian language Murrinh-patha (Seiss, 2012).

Several tools have been developed to model FSM. Proprietary tools like the Xerox Finite-State Transducer (XFST) (Beesley and Karttunen, 2003), and the FSM Library from AT&T (now in OpenFST) have been widely used in the past. Open source

solutions like OpenFST (Allauzen et al., 2007), HFST (Lindén et al., 2009) and Foma (Hulden, 2009) are also employed. XFST has been used widely as an aid to grammar engineering in the LFG/XLE context (Beesley and Karttunen, 2003; Butt et al., 1999; Rahman, 2016) as part of the ParGram effort. However, I wanted to choose an open-source tool with an active status. Further, since I wanted to use the analyser for the purpose of developing grammar according to the ParGram style grammar, the analyser should produce an output compatible with the XFST format. Foma satisfies all these constraints. Therefore, I used Foma to implement the proposed Tamil Morphological Analyser and Generator (MAG).

3.3.2 Foma

Foma is a C library and a compiler that is used to develop Finite-State Transducers (FSTs) for various purposes, including the development of language applications such as a Morphological Analyser and Generator (MAG). In this section, I briefly discuss how Foma can be used to model inflectional morphology.

Developing an FST-based morphological analyser generally requires two components: 1) a list of morphs (morphotactics and lexicon); 2) alteration rules (morphophonological rules or orthographical rules) (Beesley and Karttunen, 2003; Hulden, 2009). Foma employs these two components as well. A lexicon component shows the ordering restrictions of the root and its morphs and maps them to an intermediate or final form. For instance, (14) (a) shows how a plural morpheme can be mapped to intermediate forms, where the morph *s* is used to mark the morphemes *+Noun* and *+Pl*. In these entries, ∧ is used to mark the morpheme boundary. As shown in (14) (b), this does not quite yield the final results for all constructions.

(14) (a). cat +Noun +Pl:cat∧s
(b). watch +Noun +Pl: watch∧s

This is where the second component, i.e. alteration rules, comes in. Here, I define the morphophonological or orthographic changes which take place. In other words, as a first step, I consider the generic scenario, where root forms take *s* to mark plurality, and then write alternation rules for the exceptional cases, as in (16). In Foma, I can define such a rule, as in (15), where I define, for example, an eInsertion rule for plural constructions which allows for the addition of the letter *e* in front of the plural morph *s* whenever a noun ends with the letters *ch*. In this way, the final plural form for *watch* is generated as *watches*. I can define any number of such rules and can then concatenate them and apply them to the output of the lexicon component to get the final forms.

(15) define eInsertion [..] -> e || c h _ ∧ s ;

Words of a language can be divided into different classes based on their morphophonological behaviour. In developing a morphological analyser, I needed to define different lexicon components for each identifiable word class. Each of these lexicon components is associated with different sets of alteration rules.

For instance, (16) shows a simple example with two classes of lexicon entries, where the word form 'watch' is resolved both as a noun and a verb. As in (15), the alteration rule will be applied only for Class-1, i.e. for lexicon entries with +N. When these lexicon entries are compiled as a Finite-State transducer, we will get a network shown in Figure 3.2. In this network, if we feed cat+s, we will get the analysis of cat+N+Pl. Similarly, if we feed cat+N+Pl, we will get the generation of cat+∧+s.

(16)
- Class-1:

  −cat +N +Pl:cat∧s

  −watch +N +Pl: watch∧s

- Class-2:

  −watch +V +3P +Sg: watch∧s

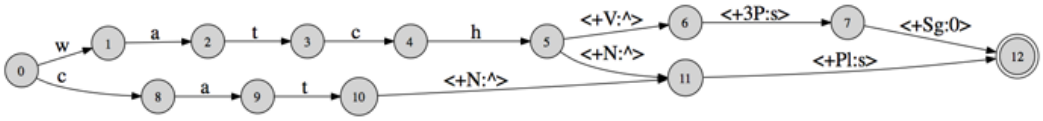

Figure 3.2: Finite-State transducer network
source: `https://fomafst.github.io/minilex.png`

Different classes can be stored as *lexc* files that can then be concatenated as necessary by the integration of alteration rules.

3.4 Literature survey

This section outlines the existing efforts in developing a morphological analyser and generator for Tamil, primarily. As you will see, although there are some attempts to develop analysers, none of them is available online or maintained or open source.

3.4.1 Approaches for developing Morphological analysers

There are rule-based, machine learning (Kumar et al., 2010a,b) and deep learning approach (Premjith et al., 2018) proposed for the development of morphological analysers for various languages, including Dravidian languages are used in literature. However,

only Premjith et al. (2018) has used a deep learning approach for a Dravidian language (Malayalam). This may be due to inadequate data available for the training of a deep learner, as stated in Bhattacharyya et al. (2019). It may also be challenging to generate the required amount of morphologically annotated content by hand to train a deep learner. On the other hand, rule-based approaches yield immediate and high-quality analyses and have been widely employed for morphologically rich languages, including South Asian languages.

3.4.2 Morphological analysers for South Asian languages

Several studies have been done on FSMs for South Asian languages. One of the earliest was Bögel et al. (2007) for Urdu, which includes a transliteration component so that the morphological analyser and generator can also be used for the structurally almost identical language, Hindi. In addition to inflectional and derivational morphology, it also tackles complex problems such as reduplication and compounding. Prasain (2011) has developed an FSM for Nepali. He first identified different classes of nouns, pronouns, verbs, adjectives, numerals, adverbs, conjunctions, postpositions, and particles, for which he then implemented a pilot morphological analyser and generator using the two-level morphology approach and XFST tool. Rahman (2016) has developed an analyser and generator for Sindhi as part of his work on grammar development for Sindhi. He also used XFST, which he then integrated into his grammar.

3.4.3 Tamil morphological analysers

Antony and Soman (2012) have carried out a survey on the state of affairs of computational morphology across Indian languages and have documented 17 efforts of morphological analysers and/or generators for Tamil. Twelve of them have been carried out before 2007, and the relevant papers, data sets and/or software are not retrievable via the Internet. The rest have been carried out since 2010. Among those five efforts, Kumar et al. (2010a,b) and Menaka et al. (2010) are available for download in binary form, yet without any data sets.

Menaka et al. (2010) and Kumar et al. (2010a) have used rule-based approaches which only perform morphological generation. On the other hand, Kumar et al. (2010b) used machine learning for the morphological analysis and generation of Tamil. They claim that the system was tested on 40,000 verbs and 30,000 nouns and that the machine learning system was trained using 130,000 verbs and 70,000 nouns from their corpus. However, no data sets, source codes, or detailed documentation is available except for a sample corpus with 270,000 tokens. The extendability of this work to aid grammar development is also questionable and needs to be researched. The authors who worked on this are no longer continuing further development of this tool.

Parameshwari (2011) has implemented a morphological analyser and generator for Tamil using a rule-based approach, which covers verbs, nouns, adjectives, pronouns, numerals and non-standard Tamil words with the use of the Apertium tool. The author claims that the system shows an accuracy of 84%. Only the research publication could be retrieved, while the associated data sets or rules are neither available in the paper nor online. Lushanthan et al. (2014) have proposed a morphological analyser and generator for Tamil, implemented using XFST. The authors have used transliteration to handle the Tamil script, given that XFST has rendering issues, although it supports Unicode internally. The authors have considered 2,000 nouns and 96 verb stems as part of their analysis and generation. They tested the system using their own data set consisting of 3,500 nouns and 500 verbs with a success rate of 78%. However, the data sets and XFST rules have not been made available.

Anna University in India developed a morphological analyser in 2001 called *Atcharam* that has recently been added to the GitHub repository.[5] It was developed for TAB (TAmil Bilingual) encoded text as a stand-alone application using Java. There is, however, no detailed technical documentation or rule set, although some data in the form of a list of words are available in the repository. These are encoded using TAB, and an attempt to convert them to Unicode was unsuccessful. Some morphological tools are also available in the Github code repository without corresponding academic publications.

Pranavan (2015)[6] has provided work on a basic morphological analyser developed as a stand-alone application using Java. However, as also claimed by the developers, it is a basic analyser that handles only 20 words with 28 conjugation forms. Yet another code repository is that by tacola-aucse.[7] This has also been developed as a stand-alone application using Java covering the analysis of verbs and nouns. However, no information about the data set or the rules developed was found. I managed to run the tool with an older version of Java, but irregular verbs such as செத்தான் (*cettān*) 'he died' do not seem to be handled, as no analysis is generated. The given analysis is confusing in some cases, especially when an out-of-vocabulary word is fed in. For instance, the analysis of சர்வேஸ்வரன் (*carvēśvaran*), a proper noun, claims that it is made up of the root சர்வே (*carvē*), the future tense marker வ் (*v*), and the past tense marker ன் (*n*). It, therefore, does not only mistake a proper noun for a verb but also provides a completely wrong analysis with two contradictory tense markers.

Additionally, the tool produces unexpected results if the text is not Unicode normalised. Finally, only one is provided in contexts where there are multiple analyses for a given word. In comparison to the other tools available, I did, however, manage to

[5] https://github.com/tacola-auceg/morpha_ta

[6] `https://github.com/Pranavan135/Tamil_Morphological_Analyzer`. There are no associated publications found.

[7] `https://github.com/tacola-aucse/Morphological-Analyzer-For-Tamil`

run this tool. However, since there is no proper documentation, it would be difficult for anyone to extend this stand-alone Java tool to make it usable for a particular need.

A Tamil Shallow Parser,[8] published in 2009, is also available, providing morphological information in addition to POS and Chunking. However, there are no papers that cover the development of the Tamil shallow parser. The authors of this tool have developed similar tools for Hindi, Telugu and Bengali, as documented in Avinesh and Karthik (2007). While the authors report results for POS and shallow parsing, they do not do so for morphological analysis. Thus, it is unclear what approach has been used for the morphological analysis since this is not discussed or evaluated. The original tool published in 2009 is available for download. The data and rules in this tool are encrypted. However, I was not successful in executing it. The first author of this tool has now re-implemented it using Python.[9] However, the new version no longer includes morphological parsing. I, therefore, used the online demo to conduct some testing myself, and the results are reported under the evaluation section.

An attempt to develop a morphological analyser for Tamil using a support vector machine (Mokanarangan et al., 2016) is also discussed in the literature. Although the authors report an accuracy of 98.73%, their system is not available, and it is not clear what data sets or analyses have been used for training and testing, as there is no data available in the paper or online.

3.5 Development of *Thamizhi*Morph

This section will discuss how I developed the proposed morphological analyser and generator in detail. I will discuss the approach, technology choices, annotation scheme, datasets, implementation steps, and evaluation.

3.5.1 The approach for developing *Thamizhi*Morph

This research retrieves what is available in terms of Tamil morphological analysers. It has shown that the existing analysers either could not be found, are incomplete, or are no longer maintained. Further, the existing analysers depend on versions of various programming languages, and the logic seems directly coded in that particular language. This nature makes these tools challenging to maintain, test or extend. Some of these applications covert and process text in ASCII, i.e. in transliterated format, and do not support Unicode encoding. Since Unicode has become the de-facto method used for text encoding, non-Unicode applications are not practically usable.

I am in the process of developing a ParGram-style computational grammar for

[8] http://ltrc.iiit.ac.in/analyzer/tamil
[9] https://github.com/avineshpvs/indic_tagger

Tamil to develop a grammar-driven parser for Tamil and datasets with morphological annotations to train a data-driven parser. The grammar-driven parser requires a morphological analyser with good precision and coverage, implemented with the use of a Finite-State morphology. On the other hand, a data-driven dependency parser requires a vast amount of morphologically annotated data. However, except for a Universal Dependencies treebank for Tamil with only 600 sentences, there is no other content available for me to train a data-driven dependency parser.

Because of all these reasons, I decided to develop a Finite-State Transducer based morphological analyser. I wanted to ensure that the morphological analyser is technology or programming language-neutral so that it can be accessed via any programming language without requiring it to be wrapped with an API (Applications Programming Interfaces). I also wanted it to be lightweight so that it could be run on any commodity hardware and be open source so that anyone could take it and extend it as needed or desired.

3.5.2 The technology stack

The application of Deep Learning in almost every NLP task has become common in the computational world. Essentially, the lack of sufficient quality data is the bottleneck for the application of deep learning approaches (Marcus, 2018) and most Indic languages, including Tamil, do not have sufficient annotated data needed for supervised machine learning.

On the other hand, Finite-State Transducers (FST) have shown proven success in the past for morphologically rich South Asian languages as discussed in Section 3.4. Since the development of a computational grammar using Lexical Functional Grammar and XLE for the ParGram project (Butt and King, 2002) is also in my project pipeline, a Finite-State morphological analyser is required. It was therefore decided to use an FST, which in addition to morphological analysis, FST has the advantage that it can also be used for morphological generation.

There are several currently available tools with which to implement an FST-based morphological analyser. These include XFST, OpenFST, and Foma. Among these, XFST has been the standard tool for developing morphological analysers. This is because an XFST-based morphological analyser can easily be integrated into a computational grammar built using XLE. However, XFST has limited support for Unicode characters, especially for complex ones such as those of Tamil.

Additionally, it is a closed-source and proprietary tool. On the other hand, Foma has support for Unicode and is open-source software that can be easily extended to web applications. For these reasons, I decided to use Foma to implement the proposed MAG.

3.5.3 Scope of annotations

I decided to capture and encode all available information that a word expresses via its different forms. Apart from POS and morphological information, I have therefore also represented morphophonological and morphosyntactic information. More importantly, in *Thamizhi*Morph, I have analysed words without conducting any pre-processing to separate multi-word tokens. This is something still to be explored in future, which will improve the coverage of *Thamizhi*Morph and solve out-of-vocabulary issues to some extent.

Among the morphological annotations I introduced, *Sandhi* annotations are important and not found in other systems. There are two types of *Sandhi* in Tamil: Internal and External. The internal *Sandhi* refers to a phonological process triggered across two morphs within a (prosodic) word, and the external *Sandhi* is triggered when such a phonological process happens at the boundary of two words. The external *Sandhi* can occur when the second word begins with one of the following consonants: க் (k), ச்(c), த் (t), ப் (p). It is triggered when further licensing conditions are met. While internal *Sandhi* is purely morphophonological in nature, external *Sandhi* is also subject to syntactic or semantic constraints. Since Tamil orthography closely reflects the phonology of the language, a *Sandhi*'s effects on the orthography must necessarily be dealt with in the development of a resource such as a computational grammar and morphological analyser. For instance, in the clause காளையைப் பிடித்தான் (*kaḷaiyaip pidittan*) 'he caught the bull', the first word *kaḷaiyaip* can be analysed as in (17), where ப் (*p*) is a *Sandhi* (SANDHI-P). In this context, the SANDHI-P comes in via assimilation following the accusative case markers, which comes about because the following word begins with a பா (paa). I have handled க் (SANDHI-K), ச் (SANDHI-C), and த் (SANDHI-T) in a similar manner in my analyses.

(17) காளையைப் (*kaḷaiyaip*)

காளை	ஐ	ப்
kaḷai	yai	p
bull	-ACC	-SANDHI-P

Tamil also has two euphonic increments (Lehmann, 1993): அன் (*an*) and இன் (*in*), which can be added onto nouns and verbs. These increments are also captured by the analysis, even though these are purely phonologically motivated.

Apart from the morphological features realised by morphs in a word, I have also included the lexically specified features shown in Table 3.5. For instance, as shown in (18), *vantāṉ* will be analysed as the lemma வா *vā* 'come' +verb|+fin|+sim|+strong| +past|+3sgm, where +fin, +sim, and +strong are lexically specified features. Since it is not always possible to extract these features orthographically, I was of the idea

that such information should still be additionally included in the analysis, as these are useful for developing applications or resources such as POS taggers and computational grammars.

(18) வந்தான் *vantāṉ* '(he) came' –> வா+verb|+fin|+sim|+strong|+past|+3sgm

Table 3.5: Lexically Specified Features

Morpheme label	**Meaning**
FIN	Finite
NONFIN	Non-finite
WEAK	Weak verb: Paramasivam (2011) discusses how a verb's weak/strong nature can be used to help determine transitivity, volitivity, affectedness and ergativity in most cases, even if he himself does not agree with this argument. I have, however, marked this information as it may be a piece of useful morphosyntactic information in the end. Note: In Tamil, weak and strong properties of words are determined based on the future tense marker.
STRONG	Strong verb
VERB	Verb
NOUN	Noun
RAT	Rational
IRRA	Irrational
SIM	Simple verb: Such verb-forms have a single verb root (as discussed in Section 3.2.2)
COMPLEX	Complex verb: Such verb-forms have more than one verbal roots, as for instance in the case of passives (also refer to the discussion in Section 3.2.2)

3.5.4 Morpheme labels

I make use of a labelling scheme developed specifically for our morphological analyser, even if there is an effort to harmonise morpheme labels across languages and tools, especially given a cross-lingual morphological transfer in mind as is, for instance, the UniMorph project (Kirov et al., 2016). UniMorph or the Universal Morphological Feature Schema "provides a set of morphological features that functions as an interlingua for inflectional morphology by defining the meaning it conveys in language-independent

terms" (Kirov et al., 2016). UniMorph, however, does not capture all the required concepts, such as the rationality feature attributed to nouns and the strong/weak nature of verbs, which are necessary for the morphological processing of Tamil. In addition, since I have developed this analyser with grammar engineering also in mind, it is always good to mark the morphological and morphosyntactic information of a single morph collectively in association with that morph. For instance, the person, number, gender and rationality values of a given noun can all be expressed by a single morph in Tamil and should be expressed accordingly. To reduce complexities in modelling, it is thus easier to mark it as a single morpheme.

In contrast, the UniMorph project proposes separate key-value pairs for all of this morphological information. While taking into account all of this, it was decided to design a scheme for morpheme labels. That could then be mapped onto other annotation schemes, as required. The list of labels used in the *Thamizhi*Morph is shown in Table 3.6.

Table 3.6: The list of labels used in the current version of *Thamizhi*Morph

Label	**Description**	**Label**	**Description**
+abl	Ablative	+loc	Locative
+acc	Accusative	+med	Medial
+adj	Adjective	+modal	Modality
+adjpart	Adjectival Participle	+mood	Mood
+adm	Admirative	+neg	Negative
+adp	Adposition	+nom	Nominative
+adv	Adverb	+nonfin	Non-finite
+appl	Applicative	+noun	Noun
+aspect	Aspect	+num	Number
+aux	Auxiliary	+oblig	Obligative
+ben	Benefactive	+opt	Optative
+cardinal	Cardinal number	+ordinal	Ordinal number
+caus	Causative	+pass	Passive
+cmpr	Comparative	+past	Past tense
+com	Commutative	+pfv	Perfective
+comp	Complimentiser	+pl	Plural
+con	Conditional	+postfix	Postposition
+conj	Conjunction	+prefix	Preposition
+cop	Copula	+pres	Present tense
+dat	Dative	+prog	Progressive
+dem	Demonstrative	+pron	Pronoun

+det	Determiner	+prox	Proximal
+dist	Distal	+prp	Purposive
+euph	Euphonic	+psp	Postposition
+excl	Exclusive	+pssd	Possessed
+fin	Finite	+refl	Reflective
+foc	Focus	+sandhic	Sandhi – C
+fut	Future	+sandhik	Sandhi – K
+gen	Genitive	+sandhip	Sandhi – P
+hab	Habitual	+sandhit	Sandhi – T
+immed	Immediate	+sconj	Subordination
+imp	Imperative	+sim	Simple
+incl	Inclusive	+soc	Sociative
+inf	Infinite	+strong	Strong verb
+inst	Instrumental	+verb	Verb
+inten	Intensifier	+vn	Verbal Noun
+lkly	Likely	+vpart	Verbal Participle
+um	A clitic	+peru	Verb ‘Get’
+idu	Verb ‘put’	+podu	Verb ‘Put’
+iru	Verb ‘be’	+poo	Verb ‘Go’
+kodu	Verb ‘with it’	+sei	Verb ‘Do’
+koduiru	Verb ‘with it – be’	+tallu	Verb ‘Push’
+koodu	Verb ‘Able’	+tolai	Verb ‘Get rid’
+mudi	Verb ‘Can’	+vaa	Verb ‘Come’
+paar	Verb ‘See’	+vai	Verb ‘Keep’
+padu	Verb ‘Suffer’	+vendu	Verb ‘Ask’
+pannu	Verb ‘Do’	+vidu	Verb ‘Leave’

Label	**Description**
+1pl	First person – Plural
+1sg	First person – Singular
+2pl	Second person – Plural
+2plh	Second person – Plural – Honorific
+2sg	Second person – Singular
+2sgh	Second person – Singular – Honorific
+3ple	Third Person – Plural – Epicene
+3plhe	Third Person – Plural – Honorific - Epicene
+3pln	Third Person – Plural – Neuter

+3sge	Third Person – Singular – Epicene
+3sgf	Third Person – Singular – Feminine
+3sgh	Third Person – Singular – Honorific
+3sghe	Third Person – Singular – Honorific – Epicene
+3sgm	Third Person – Singular – Masculine
+3sgn	Third Person – Singular – Masculine

The analyses of each word in *Thamizhi*Morph is given the following form in (19), in the system that has been developed,[10] where apart from the morphemic information or the morpheme label, the morph which corresponds to that information has also been recorded in the analysis for further future use. Additionally, each morpheme has been separated using with use of a morpheme boundary '|', similar to what is used by Beesley and Karttunen (2003) to mark term boundaries (they use 'TB'). I made this choice partly because "|" is the symbol used in Universal Dependencies (UD) to separate features.[11] (20) shows an actual output from there analyser, where *turattiṉāṉ* '(he) chased' is analysed. From the analysis, we can identify that *turuttu* is the lemma, the past tense marker is *iṉ*, and the 3sgm marker is *āṉ*. In addition, it also consists of other lexical features, listed in Table 3.5.

(19) root |+morpheme1=morph|+morpheme2=morph|...

(20) துரத்தினான் –> துரத்து|+verb|+fin|+sim|+strong|+past=இன்|+3sgm=ஆன்
turattiṉāṉ –> *turuttu* |+verb|+fin|+sim|+strong|+past=*iṉ*|+3sgm=*āṉ*
'(he) chased'

I have also taken a key decision to annotate postpositions that are suffixed to nouns. One can mark all postpositions which are suffixed to nouns as postpositions or PSP. Instead of just labelling it as PSP, I also added the actual postposition to make the analysis more meaningful and useful. Therefore, I used the following format when analysing postpositions: psptype_yyyyy where yyyyy is the actual postposition. For instance, psptype_pinnal is for nouns with the postposition *pinnal* 'after that, behind'. Similarly, postpositions taken by verbal participials are also marked in the form of vpart_yyy where 'yyy' is replaced with the corresponding postpositions.

3.6 *Thamizhi*Morph

Based on the design choices outlined in the previous section, I have developed a morphological analyser and generator for Tamil using a Finite-State approach with the

[10] `http://nlp-tools.uom.lk/thamizhi-morph/parse-sentence.php`
[11] `https://universaldependencies.org/format.html`

aid of Foma. This section outlines the architecture of *Thamizhi*Morph, including the pre-processing steps, data gathering approaches, a compilation of rules, and the development of the tool.

3.6.1 Pre-processing

Due to the nature of the Tamil Unicode encoding and input methods, a character can be represented by multiple code sequences. For instance, the letter ஹ்ரீ can be represented either as: ஹ + ◌்ர + ◌ீ or ஹ + ◌் + ◌்ர + ◌ீ. I have therefore developed a script[12] that is able to convert all input to Unicode normalised form prior to it being fed into the system for analysis or generation. That ensures that each letter and word is represented uniquely.

3.6.2 Compilation of Lexicons

Lexicons for Tamil verbs, nouns, and other particles have been compiled from various sources, including books, a dictionary, and corpora, as outlined below. The words have been classified on the basis of the paradigms discussed in Sections 3.2.2 and 3.2.1. Adjectives, adverbs, and other particles, such as conjunctions, have been compiled as separate lists.

3.6.2.1 A lexicon of verbs

A lexicon of 3,300 lemmata of Tamil verbs has been compiled from the following two verified sources:

1. Ramakrishnan (2014) has identified 369 of the most frequently used verbs in Modern Tamil. This analysis is based on a corpus of 7 million tokens compiled from the web and has taken into account expert advice on linguistic matters. This list has been included in the contemporary Tamil dictionary Cre-A (Ramakrishnan, 2014).

2. Irākavaiyaṅkār (1958) has surveyed the Tamil classic literature up until 1958, where he identified 3,124 lemmas, and categorised these into 12 classes as per the classification proposed by Graul (1855) and Sithiraputhiran (2004). Some of these forms are not used in the contemporary language. Yet, since analysing these verbs is necessary to also process historical Tamil texts, the entire list has been used for the development of the FSM.

A lexicon of complex verbs has been manually constructed by joining the infinitival form or verbal participial form of verbal roots together with secondary verbs (Boologarambai,

[12] `https://github.com/sarves/thamizhi-preprocessor`

1986) as identified in Section 3.2.2 and Table 3.3. For instance, all the modal complex verb constructions are formed by joining the infinitival form of the verb together with a modal auxiliary verb, as in (21) (a). Similarly, aspectual markers are always joined with a verbal participial form of the root verb, as in (21) (b).

(21)

(a).

செய்ய	+ வேண்டும்	= செய்யவேண்டும்
(*seiy-a*)	+ (*vendum*)	= (*seiyavendum*)
do-INF MARKER	+ want.FUT	= do.INF.want.FUT

'(will) want to do'

(b).

செய்து	+ முடித்தான்	= செய்துமுடித்தான்
(*seiy-tu*)	+ (*muditttaan*)	= (*seiytumuditttaan*)
do-VPART MARKER	+ finish.PAST.3SM	= do.VPART.finish.PAST.3SM

'(he) did finish'

3.6.2.2 A lexicon of nouns

A lexicon of 80,000 Tamil nouns was collected from online databases, glossaries, and various datasets. An initial level of cleaning was conducted to ensure that these are proper words and that they adhere to Tamil orthography and morphology. Because if a word itself is wrong, then the lexicon will be polluted. To ensure this, I developed a Python script[13] to filter out words that are unacceptable according to the Tamil grammar (Thesikar, 1957) and its current usage. According to traditional Tamil grammar, a word cannot for instance start with the letters ர (*r*) and ற (*ṟ*), yet these are widely available in present Tamil usage. Therefore, I also incorporated such usages, and in doing so, I amended the morphological analyser. However, a proper corpus-based study is required in this regard to understand Tamil spelling rules.

3.6.3 Meta-Morph rules

The novel concept of Meta-Morph rules that I have presented in Sarveswaran et al. (2019) has been integrated. Meta-Morph Rules are lexical rules in the form of meta-data that is fed into the development of the Foma-based morphological analyser. As discussed above, as part of the review of Tamil morphological analysers, most of the previous efforts at encoding the morphotactics have been deeply coupled with particular

[13] `https://github.com/sarves/thamizhi-preprocessor`

programming logic. Other efforts have relied on a heavy manual effort.

The definition and use of Meta-Morph rules help us focus on a language's analysis without the additional distraction of being bound by a particular programming logic. This, in turn, allows for the automation of the generation of lexical entries, which, when done manually, is not only a tedious and time-consuming task but also prone to error. This is particularly true for a language like Tamil, where each verb may display several hundred inflections. Moreover, even if a paradigm approach were to be used, it is challenging to write rules, maintain them and perform regression testing without the aid of a meta-grammar.

I initially developed a morphological analyser and generator by manually inputting all of the necessary lexical strings, a tedious task that took significant time and energy. However, this manual process has helped us understand Tamil's overall generalised morphological structure. In evaluating the progress, I found that correcting errors was complicated and time-consuming since I always had to engage with the details of the Foma specifications. The frustration with these sorts of tasks has led me to experiment with Meta-Morph Rules.

The idea was to find a way of stating the morphotactics needed to analyse and generate Tamil words in a manner that would be transparent, programming language independent and easy to maintain. As shown in Snippet-1, I hit upon a format that contains the following information: 1) The word classes to which the information applies (Line 1); 2) the inherent lexical specifications for that word, e.g. the classes involved can be finite, simple, and indicative (Line 2); 3) the order of the morphemes (Line 3); 4) particular patterns, for example, as in Line 4 where it is stated that verbs (of the classes defined in Line 1) which contain euphonic markers (the material used to fulfil phonological phrasing requirements) are constructed only with past tense verbs, and only with a specific PNG marker (e.g. pngeuph in Line 4).

Snippet 1 Snippet of Meta-Morph rules

```
1.classes=C1,C2,C3,C4,C5,C17,C18,C19
2.commonLabels=+fin+sim+ind
3.v-ind=root+tense+png
4.v-euph=root+past+euph+pngeuph
```

I found the writing of rules in the Meta-Morph format illustrated by Snippet-1 to be quick, easy and transparent. When it came to the translation of these descriptive statements into implementation, I found defining feature-value pairs using JSON[14] to be the most efficient way forward. Adding in the extra step of formulating Meta-Morph Rules coupled with the JSON knowledge base has helped significantly accelerate the development of an FSM for Tamil. Adding a lexical string or a new conjugation form

[14] https://www.json.org/

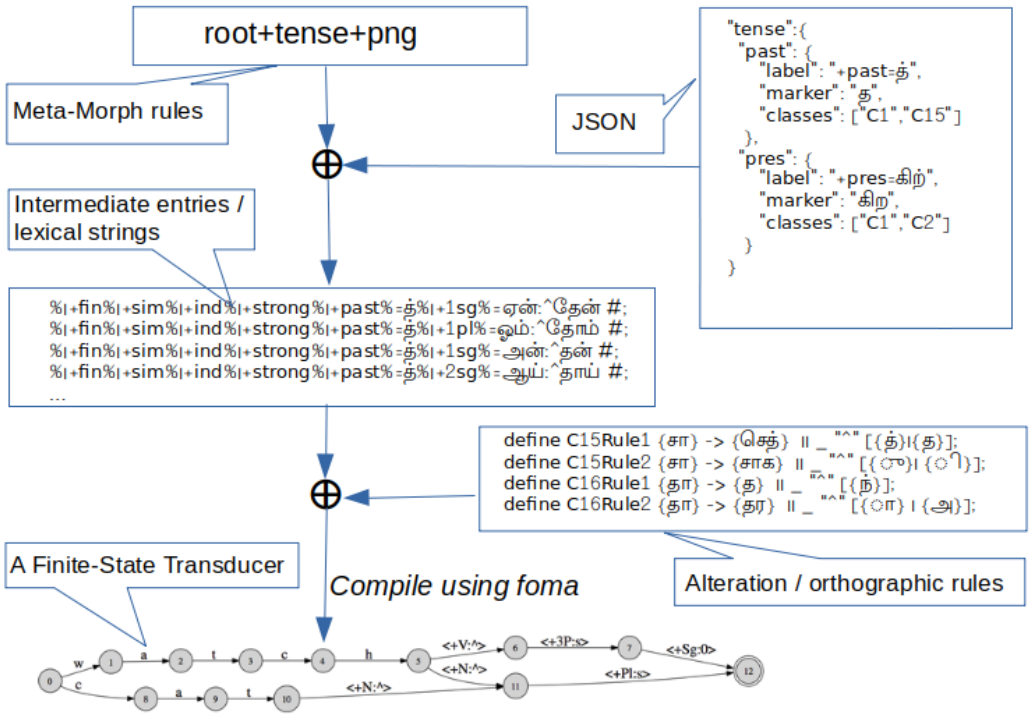

Figure 3.3: The process outline: shows how actual FST is built using Meta-Morph Rules and other components

now becomes very straightforward: All that is required is to list the classes which will take those new forms and then define a generalised rule for the formation of that word, as shown in Snippet-1. An overview of all the system components and processes is presented in Figure 3.3.

The JSON files contain detailed morphophonological and orthographic information about the values for the labels in the Meta-Morph Rules. As shown in Snippet-2, data are stored in the JSON files as key-value pairs, which are also human-readable. In addition to labels, values corresponding to each morph are stored in the JSON files, as shown in Snippet-2. For instance, tense consists of the values past, future (fut) and present (pres). These values can be further specified, as demonstrated in the past tense, where two different possibilities are provided. This information becomes part of the lexical analysis. The data structure provides desirable flexibility for defining different tense markers and labels for different classes. These data can be referred to at different levels when writing the Meta-Morph Rules. For instance, as shown in Snippet-1, both the `tense` feature or the `past` feature can be referred to independently from one another. Furthermore, in case there was a mistake in the labelling or in the specification of the value of any marker, corrections can now easily be done directly in the descriptive but hierarchical JSON text file without needing to engage with the details of the FSM programming logic.

Snippet 2 Snippet of data in a JSON file

```
"tense": {
"past": {
        "past1": {
                "label":"+past=த்",
                "marker":"த",
                "classes":["C1","C15"] },
        "past2": {
                "label":"+past=ண்ட்",
                "marker":"ண்ட",
                "classes":["C2"] }
},
"pres":{
        "pres1": {
                "label":"+pres=கிற்",
                "marker":"கிற",
                "classes":["C2","C3"] }
},
"fut": {
        "fut1": {
                "label": "+fut=வ்",
                "marker": "வ",
                "classes": ["C3","C4"] }
}
}
```

The above rules and JSON entries can be written in a plain text file. For instance, Snippet-2 shows how tense labels are defined and stored in a JSON file. As shown here, there can be different past tense markers for different classes of verbs. For general cases, the tense marking can be done as shown in line 3 of Snippet-1. However, if required, a particular tense marker can also be used, as shown in line number 4 of Snippet-1.

Once the Meta-Morph Rules are finalised, they can be parsed to produce actual lexical strings that are then fed to Foma to compile an FST. A Python parser has been developed to parse these Meta-Morph Rules to generate lexical rules for Foma. A sample of a compiled Meta-Morph Rule is shown in Snippet-3. As mentioned previously, I use the pipe "|" symbol to mark morpheme boundaries. The % in Snippet-3 is used to allow us to escape special characters in the lexical string.

Snippet 3 Snippet of intermediate entries or lexical/analysis string

%|+fin %|+sim %|+ind %|+strong %|+past %= த் %|+3sgn %=அது

Apart from the generation of these intermediate entries, orthographical rules have been written for each class in the paradigm, as necessary, based on the description in Section 3.6.4. If a new class needs to be introduced, then a new set of entries needs to be added to the orthographical file. Otherwise, there is no need to touch the lexical strings or the orthographical files.

3.6.4 Orthographical rules

Writing orthographic rules is a complex task for a language such as Tamil. In most cases, the affixation of suffixes is not just a mere addition to a lemma. Rather, several orthographical changes occur during Tamil's affixation process due to grammatical and phonological reasons. These complicate the process of writing orthographic rules. The following are common orthographical changes that one observes when suffixation occurs. These have been programmed to be handled by the analyser and generator:

- Morpheme/s can be suffixed to the lemma. However, when a vowel follows a consonant, these two together become composite. For instance, செய் (*cey*) 'do' + த் (*t*) 'past tense marker' + ஆன் (*aan*) 'third person, masculine, singular and rational' = செய்தான் (*ceytaan*) '(he) did', where த் (*t*) + ஆ (*aa*) together form the composite character தா (*taa*).

- A new letter is introduced in addition to the morpheme. For example நட (*naṭa*) 'walk' + த் - (*t*) 'past tense marker' + ஆன் (*aan*) 'third person, masculine, singular and rational' = நடந்தான் (*naṭantāṉ*) 'walked (he)', where a letter ந் (*n*) is introduced. Some researchers consider this as a *Sandhi* letter, but some modern linguists consider ந்த் (*nt*) to function as the past tense marker (Nuhman, 1999).

- Two letters together can form a new letter during suffixation. For example கொள் (*koḷ*) 'take' + ட் (*ṭ*) 'past tense marker'+ ஆன் (*aan*) 'third person, masculine, singular and rational' = கொண்டான் (*koṇṭaan)* 'he took', where ள் (*ḷ*) becomes ண் (*ṇ*).

- A new morpheme, referred to as a euphonic morpheme, can be introduced in addition to the required morphemes. For instance in the word செய்தனம் (*ceythanam*) '(we) did', செய் (*cey*) 'do' + த் (t) 'past tense marker' + (அன் *an* (euphonic

marker)) + அம் (*am*), which expresses the following information: first person, neuter, plural and rational.

- Irregular verbs in Tamil may undergo a complete change in their surface form when they are conjugated. தா (*taa*) 'give' takes different forms such as தா / தரு / த (*taa/taru/ta*). For instance, in past tense forms, the lemma becomes த (*ta*), and in present and future tense forms it becomes தரு (*taru*). The imperative form is தா (*taa*). The variations in forms yield: தந்தான் (*tantaan*) 'gave(he)', தருவான் (*taruvaan*) '(he) will give', and தா (*taa*) 'give'.

- A consonantal glide may be introduced when two consecutive vowels exist. Tamil has two such consonantal glides: ய் (*y*) and வ் (*v*).

So far, I have discussed how the analyser is planned and developed. However, the analyser can only analyse words included in the lexicon. Therefore, I have also implemented a morphological guesser to analyse new words that are not in the lexicon.

3.6.5 Morphological guesser

I have also implemented a guesser to analyse nouns, verbs, adjectives, and adverbs that are not part of the lexicons compiled. To do so, I have identified common suffixes along with their corresponding analyses to develop this guesser. If the guesser does not match any suffixes which I have listed, then it will recognise the word as a noun with a nominative case. This guesser is a valuable component of tackling out-of-vocabulary problems.

3.7 Evaluation and Discussion

There are currently no benchmark data sets available for Tamil to evaluate language processing applications, including morphological analysers. Researchers tend to use their own data sets to evaluate and report results. Here I report on two experiments to be individually discussed in more detail in the sections to follow. In the first instance, I made use of texts from an elementary Tamil textbook that is part of the Sri Lankan school curriculum. This contains 612 unique words and comprises words that have a good sample coverage of different types of POS, compound words, and foreign words. With this data set, I have conducted a comparative evaluation of *Thamizhi*Morph and the IIIT's Tamil shallow parser, which I discuss below. As part of my second experiment, I made use of a Tamil text from UD v2.5 to conduct a detailed error analysis.

3.7.1 Comparing *Thamizhi*Morph and the IIIT Parser

I have not successfully run IIIT's shallow parser locally, and the version re-implemented using Python does not have a morphological analysis. I had to rely instead on the available web interface.[15] Since it is a shallow parser, the input has to be a phrase, at least. I also found that the analyser does not provide a morphological analysis based on the context of a given word. It just generates all analyses in Shakti Standard Format (SSF) annotation (Bharati et al., 2007).

Home தம்பி வந்தான் Instructions

```
<Sentence id="1">
1       ((          VGNF    <fs af='வா,v,m,sg,3,,ந்த்,nw_AnY' head="வந்தான்"
                            tense="PAST" paradigm="v25" finite="Y">
1.1     வந்தான்     VM      <fs af='வா,v,m,sg,3,,ந்த்,nw_AnY' name="வந்தான்"
                            tense="PAST" paradigm="v25" finite="Y">
        ))
2       ((          NP
2.1     உந்க்       NN
        ))
</Sentence>
```

Figure 3.4: A sample output from the IIIT shallow parser

For instance, Figure 3.4 shows the analysis for தம்பி வந்தான் - *(thambi vanthan)* - brother.NOM come.PAST.3RSM - 'younger brother came'. For some reason, the analysis produced by the IIIT parser always results in the first word missing. Figure 3.5 shows the analysis for the same clause from *Thamizhi*Morph. There we see how in addition to the regular analysis of 'brother', the morphological analyser also provides a guessed version via the guesser integrated for nouns.

தம்பி வந்தான்	
தம்பி	தம்பி தம்பி|+noun|+nom தம்பி தம்பி+guess|+noun|+sg|+nom
வந்தான்	வந்தான் வா|+fin|+sim|+ind|+strong|+past=ந்த்|+3sgm=ஆன்

Figure 3.5: A sample output from *Thamizhi*Morph

I passed the 612 words taken from the elementary school Tamil text to the IIIT shallow parser, using a script I developed, and to *Thamizhi*Morph. The results are shown in Table 3.7. As shown, out of 612 words, the IIIT parser analysed 585 words,

[15] `http://ltrc.iiit.ac.in/analyzer/tamil`

and *Thamizhi*Morph analysed 571 words. Table 3.7 also shows how many of those analyses are correct analyses and what percentage of lemmas are correctly predicted. While *Thamizhi*Morph always gives a correct analysis, the IIIT parser fails to guess the lemma associated with 12 words.

Table 3.7: *Thamizhi*Morph vs. IIIT Tamil Shallow parser

	IIIT Parser	***Thamizhi*Morph**
Analysis found	95.6%	93.3%
Right analysis found (out of successful analyses)	96.2%	100%
Right Lemma prediction (out of successful analyses)	94.4%	97.9%

The error analysis showed that the IIIT parser produces incorrect morphs for some verbs. For instance, in the verb தெரியும் (*teriyum*) 'will know', the future tense is marked as -ப்ப்- (*-pp-*), although it is zero marked. Another source of error production is the dropping of some characters or the introduction of new characters. Examples of such include the roots கொடுப்பீர்கள் (*koṭuppeerkaḷ*) '(you) will give' and ஆச்சரியப்பட்டது (*aaccariyappaṭṭatu*) '(it) was surprised' become *கொடுபீர் *(koṭupeer*) and *ஆச்சரியம்பட்டா (*aaccariyampaṭṭaa*) instead of கொடு (*koḍu*) 'give' and ஆச்சரியப்படு (*aaccariyappaḍu*) 'be surprised' respectively. The analyses that are produced are meaningless. In addition, the analyser always provides one analysis for a given word, irrespective of the context. That is, it always resolves potential ambiguity in only one way. I conclude that even though this tool is available and accessible through the Internet, it still does not show a sufficiently good performance.

The errors found with *Thamizhi*Morph are instances that have to do with either out-of-vocabulary items or derivational formations that have not yet been included in the FSM, yielding items that are unknown to the analyser.

The out-of-vocabulary errors are easily remedied, as that merely requires me to add the missing vocabulary to the lexicon table which I have created. The derivational formations are complex, and I have only just begun adding derivations on a case-by-case basis. I must reiterate, however, that the initial focus of this work is the development of a morphological analyser for the inflectional morphology and not derivational morphology. As things stand, it is not surprising that *Thamizhi*Morph cannot as yet analyse instances of derivational morphology; nevertheless, with the effort in producing a MAG that handles Tamil inflected morphology quite robustly, I do have the flexibility to now proceed with the necessary corrections and extensions.

3.7.2 Evaluating *Thamizhi*Morph using UD Tamil Treebank

In the second experiment, I used the text in the available Tamil Universal Dependencies Treebank v2.5 to evaluate *Thamizhi*Morph. The treebank, in total, consists of 8,635 tokens from 600 sentences. There are 3,567 unique words before tokenisation, which is increased to 4,055 tokens after multi-word tokenisation. Because there are inaccuracies in the multi-word annotations and the UD annotations, I decided to work with the list of items prior to tokenisation, i.e., the 3,567 unique words. Of these, *Thamizhi*Morph successfully analysed 3,023 words, i.e. 84.7% of the words, but failed to analyse 544 words. The following are the reasons for the errors:

- 240 words were not present in the lexicon I used. Most of these are due to the differences between Indian Tamil and Sri Lankan Tamil. டாஸ்மாக் (*ṭaasmaak*), and டிஜிபி (*DGP*) are instances of Indian Tamil words that are missing in my lexicon. Apart from that, other missing words were acronyms and proper names.

- there are several spelling mistakes in the treebank. Sometimes, the same word has been written in multiple wrong forms. For instance, ஆமதாபாத் (*aamataabaat*) 'Ahmedabad' is spelt incorrectly. *கம்யூனிஸட் vs கம்யூனிஸ்ட் *(camyuunisaṭ* vs *camyuunisṭ*) 'communist' are two forms, where one is correct and other one is wrong.

- Noun+Verb compound constructions were not analysed correctly. An example is: கவலைப்படாதீர்கள் (*kavalappaṭaateerkaḷ*) '(you) do not worry'. I have now added this to the lexicon in terms of a complex root, adding the root N+V as a lexical item.

- the analyser and guesser did not correctly parse some Noun+Noun compound constructions. An example of this is இரண்டரை (*iraṇṭarai*) 'two and a half'. These have now been added as items to the lexicon to be able to analyse these constructions.

In working with the UD Tamil treebank, I identified numerous annotation problems with it. One necessary adjustment is the need to extend the current UD morphological labels to reflect all of the actual morphological information I have incorporated. This incorporation can be done via the language-specific features proposed in the UD guidelines.[16] For instance, I found that the current version of the UD morphological feature inventory does not have labels to mark rationality, euphonic markers, and *Sandhi* effects.

[16] `https://universaldependencies.org/docs/ext-feat-index.html`

3.8 Morphological generation

As discussed in Section 3.3.1, Finite-State Transducers have two tapes, namely input and output. Therefore, when one feeds in a word, an FST provides an analysis, as discussed in the previous sections. Similarly, it also gives the word or surface form when feeding a lemma form along with its morphemes from the receiving end of the network. As in Figure 3.2, the tape has values separated with a 'colon', and the left side of colons captures the analysis, and the right side captures the generation. Null morphs are represented by '0' in the mentioned figure.

*Thamizhi*Morph can also generate surface forms when a lemma and the morpheme labels are provided. Using the current version of *Thamizhi*Morph, I could generate more than 1.4+ million simple verbs, and 54+ million complex verbs (verbs with markers like mood, aspect *etc.*). I have uploaded these generated words to Kaggle[17] under the Creative Commons license[18] so that anyone in the world can use them. These lists are useful for several natural language processing tasks, including building a language lexicon, spell checkers, and machine translation. The number of words in the generation process depends on the size of the lexicon and the number of conjugational forms each class of words takes. Some of these generated words may never occur in the language, yet they still constitute valid constructions according to the morphological rules.

There is one limitation that is, however, unavoidable during the generation process. I have used guessers to guess the analysis of Tamil verbs and nouns. This means that words that do not exist in the lexicon can also be guessed and analysed. However, these guessers are not developed to do the same for a generation. Therefore, at times even if a word is analysed properly, it may not be generated.

3.9 *Thamizhi*Morph to UD tagging

I have incorporated a feature to the current version of *Thamizhi*Morph to populate morphological analysis in the CoNLL-U format,[19] which is used in the Universal Dependencies treebanks. I did this to generate morphological features for the Tamil UD treebank, which I used to train a data-driven parser outlined in Chapter 6. In UD, morphological analyses are represented as key-value pairs. In order to do this, first, I created a mapping from *Thamizhi*Morph annotation to the Universal Dependencies key-value pair annotation, which is shown in Table 3.8 and 3.9. Table 3.8 lists the labels where there was a one-to-one mapping between *Thamizhi*Morph and Universal Dependencies, while Table 3.9 shows labels where a single *Thamizhi*Morph has, on the other

[17] `https://www.kaggle.com/sarves/tamilverbs`
[18] `https://creativecommons.org/licenses/by-nc/4.0/`
[19] `https://universaldependencies.org/docs/format.html`

hand, been mapped onto multiple key-value pairs. I believe that this extension I have developed could be a useful resource for the creators of the Tamil Universal Dependencies treebank as well as for other researchers who are more interested in cross-lingual studies.

Table 3.8: One-to-One mappings between *Thamizhi*Morph (TM) and Universal Dependencies (UD) labels

TM	**UD**	**TM**	**UD**
+adp	AdpType=Post	+um	Clitic=um
+postfix	AdpType=Post	+excl	Clusivity=Ex
+psp	AdpType=Post	+incl	Clusivity=In
+prefix	AdpType=Prep	+med	Deixis=Med
+hab	Aspect=Hab	+prox	Deixis=Prox
+imp	Aspect=Imp	+dist	Deixis=Remt
+pfv	Aspect=Pref	+adm	Mood=Adm
+prog	Aspect=Prog	+con	Mood=Cnd
+idu	AuxType=Idu	+imp	Mood=Imp
+iru	AuxType=Iru	+jus	Mood=Jus
+kodu	AuxType=Kodu	+oblig	Mood=Nec
+koduiru	AuxType=Kondiru	+inten	Mood=Prp
+koodu	AuxType=Koodu	+prp	Mood=Prp
+mudi	AuxType=Mudi	+pl	Number=Plur
+paar	AuxType=Paar	+cardinal	NumType=Card
+padu	AuxType=Padu	+fraction	NumType=Frac
+pannu	AuxType=Pannu	+ordinal	NumType=Ord
+peru	AuxType=Peru	+neg	Polarity=Neg
+podu	AuxType=Podu	+pssd	Poss=Yes
+poo	AuxType=Poo	+dem	PronType=Dem
+sei	AuxType=Sei	+pron	PronType=Prs
+tolai	AuxType=Tolai	+refl	Reflex=yes
+vaa	AuxType=Vaa	+sandhic	Sandhi=C
+vai	AuxType=Vai	+sandhik	Sandhi=K
+vendu	AuxType=Vendu	+sandhip	Sandhi=P
+vidu	AuxType=Vidu	+sandhit	Sandhi=T
+abl	Case=Abl	+fut	Tense=Fut
+acc	Case=Acc	+immed	Tense=immed
+ben	Case=Ben	+past	Tense=Past
+caus	Case=Cau	+pres	Tense=Pres

+caus	Case=Caus	+appl	Valency=Appl
+cmpr	Case=Cmp	+vpart	VerbForm=Conv
+dat	Case=Dat	+fin	VerbForm=Fin
+gen	Case=Gen	+inf	VerbForm=Inf
+inst	Case=Ins	+adjpart	VerbForm=Part
+loc	Case=Loc	+foc	Voice=Focus
+nom	Case=Nom	+pass	Voice=Pass
+opt	Case=Opt		
+soc	Case=Soc		

Table 3.9: One-to-Many mappings between *Thamizhi*Morph (TM) and Universal Dependencies (UD) features

***Thamizhi*Morph**	**Universal Dependencies features**
+vn	PronType=Prs|VerbForm=Vnoun
+1pl	Person=1|Number=Plur
+1sg	Person=1|Number=Sing
+1sg	Person=1|Number=Sing
+2pl	Person=2|Number=Plur
+2pl	Person=2|Number=Plur
+2plh	Person=2|Number=Plur|Polite=Form
+2sg	Person=2|Number=Sing
+2sgf	Person=2|Number=Sing|Gender=Fem
+2sgm	Person=2|Number=Sing|Gender=Masc
+2sgh	Person=2|Number=Sing|Polite=Form
+3pln	Person=3|Number=Plur
+3ple	Person=3|Number=Plur|Gender=Com
+3plhe	Person=3|Number=Plur|Polite=Form|Gender=Com
+3sge	Person=3|Number=Sing|Gender=Com
+3sgf	Person=3|Number=Sing|Gender=Fem
+3sgm	Person=3|Number=Sing|Gender=Masc
+3sgn	Person=3|Number=Sing|Gender=Neut
+3sgh	Person=3|Number=Sing|Polite=Form
+3sghe	Person=3|Number=Sing|Polite=Form|Gender=Com

Additionally, some single labels in *Thamizhi*Morph have been mapped onto several key-value pairs in UD, as illustrated in Table 3.9. There were some other instances where the annotations found in *Thamizhi*Morph do not exist in UD morph. In order to find suitable labels, I consulted the UniMorph (Kirov et al., 2016) project. For

features related to *Sandhi*, and rationality, UniMorph does not provide any definition. I have therefore introduced new labels for these. In addition, I have also introduced values for features for AuxType, as I could not find the correct value for some of the auxiliary[20] types which I have identified from the feature list specified by UniMorph. To associate items with this particular feature, I have specified the root forms of auxiliaries as AuxType, for instance, 'AuxType=kondiru', where *kondiru* is an auxiliary verb.

3.10 Conclusion

In this chapter, I have described the design and development of a Tamil Morphological Analyser cum Generator, *Thamizhi*Morph, along with all the tools and the relevant ways with which such are supported.[21][22]

The Finite-State Transducer (FST) models that I have compiled can be used for language processing applications. I am currently using them in the context of Tamil grammar development and treebank creation.

While no benchmark resources exist for evaluating NLP applications developed for Tamil, I have designed two evaluation experiments to test the coverage and accuracy of *Thamizhi*Morph. The results are quite robust in that identified errors are either due to out-of-vocabulary items or derivational formations that have not as yet been implemented.

Overall, I have made the following contributions with respect to the development of a Tamil morphological analyser cum generator:

1. created a robust rule-based morphological analyser cum generator using an open-source framework. I have made this available as a Python library called thamizhilip (Thamizhi Linguistic Processing Library),[23][24] which can be easily set up and used for developing other language processing applications, including grammars. I have also published this as a web-based application.[25]

2. introduced a novel concept of Meta-Morph Rules, which helps researchers to focus on the analysis of a language and make the development and maintenance of a morphological analyser efficient and effective by reducing manual effort.

[20] In morphological analysis, the term 'auxiliary' verb is used in a very broad sense as used in Nuhman (1999). However, I am aware that not all the verbs that are marked as auxiliary are not always auxiliary verbs. These could be light verbs as well. This level of analysis is to be taken up in more detail later in the syntactic analysis.

[21] `http://nlp-tools.uom.lk/thamizhi-morph/`

[22] `https://github.com/sarves/thamizhi-morph`

[23] `https://pypi.org/project/thamizhilip`

[24] `https://sarves.github.io/thamizhilip`

[25] `http://nlp-tools.uom.lk/thamizhi-morph/`

3. devised a mapping between the *Thamizhi*Morph annotation scheme and the Universal Dependencies scheme, which is useful to generate morphological annotations for creating treebanks. In fact, I used this tool to generate morphological annotations when I created a Tamil treebank.

4. provided a detailed account of the morphology of Tamil verbs and nouns, along with respective paradigms. The incorporation of *Sandhi* and rationality into the morphological analysis of Tamil is a new phenomenon.

5. compiled noun and verb lexicons from various sources and through various processes. These are also made available for reuse.

6. generated millions of nouns and verbs and have published them in Kaggle[26] for other researchers and engineers to use and develop machine learning applications.

3.11 Future work

There are a number of ways in which the current version of *Thamizhi*Morph can be improved and extended. Most importantly, the following improvements can be made:

- I have found a database of close to 10 million words and their frequencies (Kunchukuttan et al., 2020) extracted from a large corpus of text. This lexicon can be used to improve the coverage of morphological analysis and generation.
- One can go through the list of words from the above list, identify more conjugational forms, and add these to *Thamizhi*Morph.

Linguistically, deep analytical studies are required. For instance, one can explore the following to start with:

- I have been labelling postpositions with their actual forms. This can be improved further if the more in-depth study of postpositions could render better sorts of labels. The same follows for auxiliary verbs.
- most of the words which have been identified as adjectives and adverbs are actually derived from nouns. Those words are, however, just being marked as either adjectives or adverbs. One can go deeper into this dimension and explore how the actual analysis can be annotated. For instance:
  - வேகமான கார்
    (vēkamāṉa kār)
    'fast car'

[26] https://www.kaggle.com/sarves/tamilverbs

where, *vēkamāṉa* = *vēkam* 'speed' + *āṉa* '-*āṉa* renders an adjective from a noun'

– வேகமாக வந்தது
(*vēkamāka vantatu*)
'came fast (it)'
where, *vēkamāka* = *vēkam* 'speed' + *āka*'-*āka* renders an adverb from a noun'

The current version of *Thamizhi*Morph does handle derivations (although I have included some derivational analyses). For instance, verbal nouns are derived from verbs. One can study this in more detail and incorporate more derivations while exploring how a derivational morphology tool can be built. This development will constitute a more complex task.

Apart from such consideration, I have done some additional experiments on context-aware morphological analyses and machine learning applications for morphological analysis; however, more is to be done on these topics as a way forward. At present, *Thamizhi*Morph will give all possible analyses when available. There are, however, two challenges associated with the disambiguation of analyses, and these are: 1) the identification of the correct Part of Speech and 2) the identification of the correct analyses based on the Part of Speech.

I have attempted a rule-based approach to do this context-aware tagging. In order to identify the Part of Speech, I made use of the *Thamizhi*POSt POS tagger outlined in Chapter 2. Since ThamizhiPOSt is a context-aware POS tagger, it can give the POS based on the given context. However, disambiguation was not a feasible possibility solely with the knowledge of POS and morphological analyses. I have had to write some syntactic rules to resolve that. What this means, therefore, is that a syntactic parser is required in order to perform this sort of disambiguation. The free-word order nature of the syntax makes this a more challenging task. Fortunately, the syntactic parser I developed captures the context and the output on which the analysis is based.

I have also tried to develop a morphological analyser using a machine learning approach. In order to do this, I have attempted an encoder-decoder model and tackled the morphological analysis as though it were a translation problem. However, I later came across a deep neural network-based NLP toolkit called Stanza (Qi et al., 2020) that includes a morphological analyser. Therefore, I stopped my exploration of developing a deep learning-based analyser and decided to try Stanza first. However, I could not progress further on it due to time availability. Having said that, one can still consider developing a deep learning model for morphological analysis and generation using a multilingual approach that also incorporates other Dravidian languages.

4 TREEBANKS, PARSING, AND TAMIL SYNTAX — AN OVERVIEW

This chapter discusses treebanks, parsing, and Tamil syntax, which then help understand and appreciate the following two chapters on grammar-driven parsing (Chapter 5) and data-driven parsing (Chapter 6).

4.1 Treebanks and Grammars

4.1.1 Grammar formalisms

The grammar of a language is a set of rules of how words are put together to form proper sentences or phrases. These rules can be mathematically developed or modelled; these mathematically modelled language specifications are called formalisms (Asudeh, 2006). The study of grammar has a long history; Pāṇini's grammar of Sanskrit - *Aṣṭādhyāyī* - was published more than two thousand years ago. *Aṣṭādhyāyī* is still being used to teach Sanskrit and used to develop treebanks (Bharati and Sangal, 1993). Similarly, *Tolkāppiyam* - the first Tamil grammar - also came about at least a thousand five hundred to two thousand one hundred years ago (Zvelebil, 1973).

Grammar formalisms are useful to model and document a language. More importantly, formalisms are required to create a model of a language that both humans and computers can understand. Several such formalisms are used, including Tree-Adjoining Grammar (TAG) (Joshi and Schabes, 1997), Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1982), Combinatory Categorial Grammar (CCG) (Steedman, 2000), and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994).

Phrase structure and dependency structure grammar are two popular formalisms used nowadays in which the relationship among words are studied and modelled (Jurafsky and Martin, 2017). The basic concept behind phrase structure or constituency is that groups of words may act as one unit, called a phrase, and there can be a relationship among these phrases. On the other hand, dependencies are constructed on the basis of the fact that words have grammatical functions with respect to other words in the sentence like subjects and modifiers (Nivre, 2005). Phrase structure grammar can be used to efficiently model languages that have fixed word order. However, when

it comes to relatively free word order languages such as Tamil, phrase structure grammars are expensive, as one needs to write production rules for every possible word order arrangement.

On the other hand, the significant advantage of dependency grammar is that it can deal with morphologically rich languages with a relatively free word order (Jurafsky and Martin, 2017). Therefore, dependency grammar is widely used in languages such as Tamil, which have a relatively free word order. The Lexical-Functional Grammar, as a hybrid formalism, uses the power of both phrase structure and dependency structure (for functional structure, in the LFG terminology). I have used the LFG formalism for grammar-driven parsing and the dependency formalism for data-driven parsing in my analysis and implementation.

4.1.2 Treebanks

Treebanks are a collection of texts with various levels of annotations, including Part of Speech (POS) and morpho-syntactic annotations. There are different formalisms used to mark syntactic annotations (Marcus et al., 1993; Bejček et al., 2013; Nivre et al., 2016; Kaplan and Bresnan, 1982). Among the available formalisms, the dependency grammar formalism is useful for languages such as Tamil that are morphologically rich and whose word order is relatively variable and less bound (Bharati et al., 2009). The American National Corpus (Ide and Suderman, 2004), the Brown Corpus, and the British National Corpus (BNC) (Aston and Burnard, 1997) consist only of Part of Speech annotations.

Several annotation schemes are used in the different treebanks, including the Penn Treebank Annotation (Marcus et al., 1993), the Prague Dependency Treebank (PDT) (Hajic et al., 2001) annotation and the Universal Dependency annotation (Nivre et al., 2016). The Penn Treebank (Marcus et al., 1993) consists of English text and is widely used in language processing, especially in the processing of the English language. It uses phrase-based structure grammar to construct the treebank where phrasal information is annotated. For instance, the sentence "Kannan loves Ratha" is annotated as shown in (1), according to the Penn Treebank annotation scheme (Marcus et al., 1993). This shows that sentence S has a noun phrase and a verb phrase, and the object NP — Ratha is included as part of the verb phrase.

(1) (S (NP (NNP Kannan)) (VP (VPZ loves) (NP (NNP Ratha))) (. .))

The Prague Dependency Treebank (PDT) (Hajic et al., 2001) is a dependency treebank that supports detailed annotations in three layers: morphological, analytical, and tectogrammatical. In the morphological layer, word-form, lemma and morphological tags are annotated, including person, number, and gender. Structural information,

including syntactic and dependency information, are annotated in the analytical layer. At this layer, a dependency tree is constructed. In the tectogrammatical layer, which is the innermost layer, deep structures are annotated (Hajič et al., 1999; Ramasamy, 2012). The PDT was initially created to annotate the Czech language, which is also a morphologically rich language. However, other languages have been incorporated into the PDT, e.g., the Prague Arabic Dependency Treebank (PADT) that has been developed with the Arabic text (Hajic et al., 2004).

The Infrastructure for the Exploration of Syntax and Semantics (INESS) is a Norwegian infrastructure that provides access to treebanks that are syntactically and semantically annotated (Rosén et al., 2012). This infrastructure supports and hosts all the available treebanks that exist. Further, INESS has a search, analysis, and advanced visualisation through which different treebanks can be compared and analysed. Unlike other consolidation efforts like META-SHARE[1] and CLARIN, the INESS infrastructure has a middleware that allows for the provision of a cross treebank analysis, irrespective of the type of the treebank.

4.2 Syntactic parsing: Literature and Approaches

Mapping from a string of words or sentences onto its parsed tree representation is called syntactic parsing (Jurafsky and Martin, 2017). As discussed in Chapter 1, parsing is an important task for linguistic analysis and the development of natural language applications.

4.2.1 Constituency and Dependency parsing

Creating a sentence's phrase structure or constituency structure using the phrase structure grammar is called constituency parsing. This type of parsing usually outputs a limited amount of syntactic information, such as phrasal structures and clause boundaries. In this approach, non-overlapping segments of a given sentence, such as noun phrases, verb phrases, adjective phrases, and prepositional phrases, are found. This is useful for language processing tasks such as information retrieval that may not require complete or complex structural information (Jurafsky and Martin, 2017). The phrase structure-based constituent analysis is more suitable for fixed word order languages such as English (Muralidaran and Misra Sharma, 2018), in which the placement of words matters to show the syntactic information or structure.

Dependency parsing marks the dependency among the words or tokens of a sentence. This is the widely used approach for parsing at present. The output consists of labelled links representing dependency relationships between words and phrases (Jurafsky and

[1] www.meta-share.org

Martin, 2017). Further, the dependency parser does not rely heavily on the position of words and the internal grouping of surface chunks and is, therefore, more suitable for free word order languages (Tsarfaty et al., 2010).

Apart from these two, there are hybrid approaches where a parse consists of both phrase structure and dependency information (Kübler et al., 2009). Lexical-Functional Grammar can be considered a hybrid approach since it incorporates two primary structures: constituency and functional structures. The functional structure can itself be represented as a dependency tree.

4.2.2 Grammar-based and Data-driven parsing approaches

Grammar-based and Data-driven parsing are two important approaches to parsing text (Nivre, 2005). Even though, in recent years, data-driven parsing has become state-of-the-art and is widely used for morphologically rich and free word order languages such as Czech and Turkish (Che and Zhang, 2017; Husain, 2009), grammar-based parsers are still useful for accurate and deep parsing and making high-quality treebanks in the first place.

Annotated treebanks are the primary resource for this kind of data-driven approach. However, not many Indian Languages except Hindi have annotated treebanks with a significant number of tokens. Moreover, even if treebanks are available, those are either kept closed or relatively small. There have been many efforts to develop data-driven dependency parsing for Hindi, Telugu, and Bangala through the ICON 2009 shared task contest. In the case of Tamil, if we are to employ supervised data-driven parsing, an annotated dataset must be first created. A grammar-based parser, on the other hand, does not require an annotated treebank. However, it requires a corpus of text that contains the syntactic constructions that the parser then needs to handle. This approach requires a deep understanding of the language's syntax and a good knowledge of formalisms. Formalisms like LFG and HPSG are deep linguistic formalisms that are widely used to develop grammar-based parsers. While this type of parser performs very accurately, it, however, fails when new syntactic constructions or new lexemes or words are found. Consequently, the grammar-based approach is not deemed robust compared to the data-driven approach.

4.2.3 Algorithms for dependency parsing

Two popular algorithms used in data-driven parsing are the transition-based parser and graph-based parser. Transition-based parsers are based on transition systems that consist of a set of states and the transitions between them (Kübler et al., 2009). Transition-based parsers have been implemented using a shift-reduce parsing algorithm and stacks; for instance, Stanza (Qi et al., 2020) and MaltParser (Nivre et al., 2006) are examples

of transition-based parsers. Graph-based parsing is based on the Minimum Spanning Tree algorithm, and MST parser (McDonald et al., 2005) and uuparser (Kiperwasser and Goldberg, 2016) support graph-based parsing.

Implementations of these two algorithms are available as off-the-shelf tools, as mentioned. Therefore, instead of developing these algorithms and parsing approaches from scratch, I have used these available tools with which I could implement the Tamil parser. Compared to graph-based parsing, transition-based parsing is widely used for language dependency parsing (Che and Zhang, 2017; Husain, 2009).

Projectivity in dependency parsing is an important phenomenon. According to Jurafsky and Martin (2017), an arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence, and the dependency tree is set to be projective if all the arcs that make it up are projective. However, languages will have valid constructions that lead to non-projective trees; especially such constructions are common among languages with relatively free word order. For instance, Figure 4.1 shows an example for an non-projective tree, because the arc from *flight* to its modifier *was* is non-projective since there is no path from *flight* to the intervening words *this* and *morning*. The initial implementation of transition-based parsers is known to fail when it comes to non-projective trees (Jurafsky and Martin, 2017). However, in recent times, there has been an attempt to extend the transition-based parser to handle non-projective trees as well (de Lhoneux et al., 2017b). This new approach is also implemented in a tool called uuparser (de Lhoneux et al., 2017b). Therefore, it is vital to use tools or frameworks that support non-projective trees so that constructions found in Tamil can be implemented without any problem.

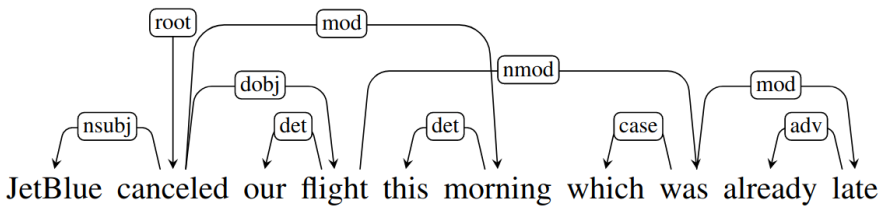

Figure 4.1: Example for a non-projective dependency tree
source: (Jurafsky and Martin, 2017)

Further, Transition-based parsing shows better accuracy with small sentences or short dependency relations. In comparison, graph-based parsing approaches can handle long dependencies well and even provide better accuracy (Jurafsky and Martin, 2017). Therefore, depending on the type of data and length of the sentences, we need to choose algorithms. The corpora I have used consist of sentences of less than 20 tokens.

4.2.4 Parsing in Tamil

Very few attempts have been made in syntactic parsing in Tamil compared to the efforts available for European languages (Ramasamy and Žabokrtský, 2011). In this section, I review the attempts that are available for Tamil:

- Ariaratnam et al. (2014) have used a rule-based approach with which to develop a shallow parser that can chunk noun phrases and verb phrases. As part of this research, a POS tagger has been developed to do the POS tagging first. On the basis of the POS-tagged data, the rules are extracted for the shallow parser. This approach has shown an f-measure of 78.3%.

- Dhivya et al. (2012) have developed a tool that can analyse a given sentence to identify clause boundaries using the MALT dependency parser. This has been mainly developed as a means with which to identify clauses in order to improve the statistical machine translation.

- Sureka et al. (2014) have developed a dependency parser for clause boundary identification that uses a similar approach specifically for machine translation. However, a Conditional Random Field-based machine learning approach is then used to construct dependency trees. They have tested this with 150 sentences and shown that the machine translation with dependency information performs better.

- A rule-based noun chunker and constituency analyser have in turn been developed by AUKBCRC and RCILTS, respectively (Rajendran, 2006). The Finite-State Machine based parser by Baskaran (1984), the syntactic parser by Kumar Shanmugam (2002), and parsing techniques by Shanmugam (2002) are reported in the paper by Rajendran (2006); the original literature could not be retrieved.

- Rule-based and Corpus-based approaches have been compared in the development of a dependency parser for the Tamil language by Ramasamy and Žabokrtský (2011). As per the results, the dependency parser, which is developed using a data-driven approach, performs better than the rule-based counterpart. The corpus-based data-driven parser has been developed using the MST and MALT parsers. A Tamil treebank based on PDT annotation (except for the tectogrammatical layer) scheme has been used to annotate the corpus manually for a data-driven approach.

- A team from IIIT Hyderabad, India, together with other research organisations in India, have developed a shallow parser that is available online.[2] However, I

[2] `http://ltrc.iiit.ac.in/analyser/tamil/`

could not find any related publications to study this tool more. Although I could not run this on my computer, I evaluated the online version and reported the results in Chapter 3.

- Dhanalakshmi et al. (2009b) have reported work on chunking using the Conditional Random Field (CRF) approach. The research identifies and segments the text into syntactically correlated word groups. In this study, authors have also shown that CRF performs better than other machine learning approaches such as Support Vector Machine and Memory-based approaches.

Based on the literature, it is evident that there are no existing parsers for Tamil available publicly for us to build upon. Further, I have outlined different parsing approaches that I will extend and use in Chapters 5 and 6.

4.3 Tamil syntax — an overview

4.3.1 Background

Until a few centuries ago, important texts in Tamil were mostly written in a poetic style, such that a non-trained person of the current generation would not be able to understand this written genre of Tamil. These texts are mostly understood through the commentaries written by scholars. At times, even these commentaries require a further simplified version in order to be understood. For instance, a scholar called *Parimēlaḻakar* wrote a commentary (Nachinarkkiniyar, 1937) for the first Tamil grammar work *Tolkāppiyam* in the 13th century. Later, many simplified commentaries were published based on *Parimēlaḻakar*'s work, due to difficulties in understanding the structure and content of those commentaries.

Although some efforts are being made to understand Tamil syntax and semantics, due to several intertwined factors that include geography, caste, religion and party politics, some work has been put aside, and others have been given importance. Moreover, some published works have not been properly circulated or catalogued. Since most of the texts written before the year 2000 use non-Unicode fonts, these texts are now not searchable over the Internet, even if some works are indeed online.

Over the years, Tamil has been in contact with several other languages. It is no longer spoken in some of the countries in which it was spoken several decades ago. Further, due to the migration that has been happening, Tamil is now spoken all over the world and has been in contact with a number of western languages as well. Apart from this array of language contact, the grammaticalisation of various words in Tamil through time makes the study more complex. There are, however, only a small number of works that relate to modern Tamil grammar. Similarly, there is no comprehensive grammar for modern Tamil written after the 13th century *naṉṉūl* grammar. It is the

derived grammar that is based on *naṉṉūl* (Thesikar, 1957) that is used as a high school Tamil textbook (Nuhman, 1999). Tamil requires a significant amount of proper linguistic study to understand and describe its grammar, especially its modern syntax. There has been an attempt to develop a modern grammar for Tamil by several scholars in Tamil Nadu, including E. Annamalai. However, no parts of the grammar have been released yet.

The following subsections highlight certain syntactic structures of Tamil that are available in corpora I am working with and which I have dealt with in this research study. I shall not attempt to go beyond syntax and formulate an overall semantic analysis for each construction mentioned here. Further, I have followed various syntactic analyses in grammar books and research papers, although these also require further linguistic exploration. For constructions such as complex predicates, light has been shed in Sarveswaran and Butt (2019) through the use of modern linguistic theories, as we could not proceed without a better understanding of these.

Since I have been using different corpora for data-driven and grammar-based parsing, not all of the outlined constructions are handled in both parsers. Tamil is a highly free word order language, although Subject-Object-Verb is the commonly used constituent order. For this reason, and for further simplification, variants other than SOV are not covered in this overview.

4.3.2 Nouns and their structures

Nouns in Tamil are marked for number and case, as discussed in Chapter 3. The number values available are singular and plural. Case[3] in Tamil has received much attention but has evolved over time. Modern Tamil grammarians propose nine cases; some are discussed in detail below. There is evidence that some case forms in Tamil are also used to mark other cases. For instance, the accusative case can sometimes be used to express the instrumental (Joseph, 1893). I will here not make reference to this further syncretic complexity. These require further linguistic explorations.

Apart from numbers and case, functional elements like adverbials, postpositions, particles, and clitics can also be attached to nouns. Adverbials are primarily of a spatial or temporal nature. While clitics can be added to nouns irrespective of their number

[3]I have not analysed whether the given case marker is "true" case marker or "just" a postposition. I just followed what is there in the literature, even though there are cases where I think more analyses are required. Anyway, this is a long-standing issue in the Tamil case system (Schiffman, 2004; Caldwell, 1998). In fact, Caldwell (1998) says the following regarding the Dravidian case system: "All case-relations are expressed by means of postpositions, or postpositional suffixes. In reality, most of the postpositions are separate words; in all the Dravidian dialects, they retain traces of their original character as auxiliary nouns. Several case signs, especially in the more cultivated dialects, have lost the faculty of separate existence and can only be treated now as case terminations. However, there is no reason to doubt that they are all postpositional nouns originally."

and case marking, particles can be attached to nouns if they are in the nominative, accusative or dative case. Adverbials and postpositions are attached to nouns if they are in the nominative or the dative case.

4.3.2.1 Nominative case

Syntactically unmarked, bare nouns are understood to be nominative-marked. These may function as a subject, predicate, subject-complement, objective-complement, or object in Tamil. Except for dative subjects, subjects in nominative cases usually necessitate subject-verb agreement, where the verb shows agreement with the nominative case marked nominal in person, number, gender (and rationality).

(2)

கண்ணன்	ஒரு	மாணவன்
kaṇṇan̲	oru	māṇavan̲
Kannan.NOM	a	student.NOM.MASC

'Kannan is a student.'

(3)

குமார்	தலைவன்	ஆனான்
kumār	talaivan̲	āṉāṉ̲
Kumar.NOM	leader.NOM.MASC	become.PAST

'Kumar became a leader.'

Example (2) is a nominal predicative construction. Such constructions are commonly used in Tamil, where the linking verb is dropped in the present tense constructions. However, for other tenses, a copula *āna* 'become' is introduced to carry tenses, as in (3). The other variant of this is shown in (4) where *āka* — an adverbial suffix — is added to the nominal predicate in (3). Although (3) and (4) are structurally different, I could not quantify it; it requires a linguistic exploration. I have glossed *āka* in (4) as ADV, because, *āka* is an adverbial suffix that is used to convert nouns adverbs.

(4)

குமார்	தலைவனாக	ஆனான்
kumār	talaivan̲āka	āṉāṉ̲
Kumar.NOM	leader.MASC.ADV	become.PAST

'Kumar became the leader.'

(5)

குமார்	இராமனைத்	தலைவன்	ஆக்கினான்
kumār	irāmaṉait	talaivan̲	ākkiṉāṉ̲
Kumar.NOM	Raman.ACC.SANDHI-T	leader.NOM	make.PAST.3SMR

'Kumar made Raman a leader.'

In N+V predication, N is always in the nominative case. For instance, (4) is a resultative structure with an N+V predicate, where N is in the nominative case.

4.3.2.2 Accusative case

The case marker *-ai* is used to mark the accusative case in Tamil. Like other case markers, this is also added to the nominal stem, oblique stem,[4] or a pluralised noun.

Tamil has differential object marking, where objects are marked with and without an accusative case on the basis of rationality. In contrast, it is obligatory to have the accusative case marking for rational entities as shown in (5),[5] marking is optional for irrational entities as shown in (6), and adding an accusative case marker onto these irrational entities ends up denoting definitiveness as in (7) (Lehmann, 1993). However, if the accusative case is not overtly marked, as in (6), the order of the object becomes restricted in relation to other constituents. Basically, the object has to follow the subject; otherwise, the sentence becomes ambiguous.

(6)

குமார்	பந்து	அடித்தான்
kumār	pantu	aṭittāṉ
Kumar.NOM	ball.NOM	hit.PAST.3SMR

'Kumar hit a ball.'

(7)

குமார்	பந்தை	அடித்தான்
kumār	pantai	aṭittāṉ
Kumar.NOM	ball.ACC	hit.PAST.3SMR

'Kumar hit the ball.'

4.3.2.3 Dative case

The dative case marked by உக்கு *-ukku* has many different functions in Tamil; Lehmann (1993) lists nine different functions. However, in my corpora, I found the following four different functions:

1. Indirect objects

Ditransitive verbs in Tamil take indirect objects marked with dative. Unlike the accusative case, the case suffix is always added irrespective of rationality.

(8)

இது	அரசுக்கு	ஒரு	சவால்	அல்ல
itu	aracukku	oru	cavāl	alla
this	state.DAT	a	challenge.NOM	not

'This is not a challenge to the state.'

[4]Some nouns take a suffix called oblique suffix, which allows the root to take other suffixes. More linguistic exploration is required in order to understand whether these oblique suffixes have or have had other functions

[5]There are a few exceptional instances where rational entities do not take the accusative case marker when functioning as an object (Lehmann, 1993). However, no such instances were found in our corpora

2. Benefactive case

On top of *-ukku*, a suffix *-āka* is added to mark the benefactive case in Tamil. Although this can be considered a separate construction type, since this is somewhat related to the dative marker, I have included benefactive under dative.

(9)

நாம்	நாட்டுக்காக	சேவை	செய்யும்	குழு
nām	nāṭṭukkāka	cēvai	ceyyum	kuḻu
we.NOM	country.BEN	service.NOM	do.FUT.REL	group.NOM

'We are a group that serves the country.'

3. Dative subjects

Dative subjects in Tamil require linguistic exploration. While some scholars consider dative subjects as indirect objects, others, such as Mohanan and Mohanan (1990), and Pappuswamy (2005), accept the concept of dative subjects. This phenomenon is found in other languages as well (Butt et al., 2006). I followed the latter sort of analysis and analysed words that are marked with the dative case that also display subject properties as dative subjects. However, unlike nominative subjects, dative subjects show mixed behaviours. They, for instance, do not show all the subject properties shown by a nominative subject, including the person-number-gender agreement with the verb. Verbs with dative objects marked for third-person, neuter-gender, and singular — are also referred to as default agreement — irrespective of subject's number, gender, rationality, and person. This default agreement is marked using உம் *-um*, as shown in (10).

(10)

எனக்கு	சிங்களம்	தெரியும்
eṉakku	ciṅkaḷam	teriy-um
I.DAT	Sinhala.NOM	know.FUT-3SN

'I know Sinhala.'

4. Goal marker

-ukku is also used to mark the goal argument of motion verbs in Tamil, as shown in (11). This is analysed as an oblique argument in my analyses.

(11)

அண்ணா	கண்டிக்குச்	சென்றான்
aṇṇā	kaṇṭikkuc	ceṉṟāṉ
elder-brother.NOM	kandy.DAT.SANDHI-C	go.PAST.3SMR

'Elder brother went to Kandy.'

4.3.2.4 Instrumental case

The instrumental case is marked by *-aal* to show instrumentation as in (12). In addition, the instrumentation can also be marked via the use of the postposition மூலம் *mūlam*

'with which', which may or may not be suffixed to the root.

Apart from this simple use-case, the instrumental case is also used to mark the agent in passive constructions, as shown in (13), and to form conditional verbal clauses, as discussed later in this chapter.

(12)

பையன்	சாவியால்	கதவைத்	திறந்தான்
paiyan̲	cāviyāl	katavait	tiṟantān̲
boy.NOM	key.INS	door.ACC.SANDHI-T	open.PAST.2SMR

'A boy opened the door with a key.'

(13)

அமைச்சரவையால்	புதிய	தீர்மானங்கள்	முன்னெடுக்கப்பட்டன
amaiccaravaiyāl	putiya	tīrmāṉaṅkaḷ	mun̲n̲eṭukkappaṭṭaṉa
cabinet.INS	new	resolutions.NOM	forward-take.PASS.PAST.3PLN

'New resolutions were taken-forward by the Cabinet.'

4.3.2.5 Locative case

In Modern Tamil, the *-il* marker is used to mark the location in space and time, and mode on irrational and rational nouns (Lehmann, 1993). For instance, (14) shows how *-il* is used to mark the location in space. *-iṭam* is another marker used to mark locative cases only on rational nouns. Lehmann (1993) claims that *-iṭam* expresses the goal of motion, transaction source, emotion target, and temporary possession. For instance, my corpora consist (15), which is an example of a goal of motion of speech.

(14)

குமார்	கூட்டத்தில்	பேசினான்
kumār	kūṭṭattil	pēciṉān̲
Kumar.NOM	meeting.LOC	speak.PAST.3SMR

'Kumar spoke at the meeting.'

(15)

குமார்	கண்ணனிடம்	பேசினான்
kumār	kaṇṇaṉiṭam	pēciṉān̲
Kumar.NOM	Kannan.LOC	speak.PAST.3SMR

'Kumar spoke to Kannan.'

4.3.2.6 Ablative case

-iruntu is used to mark ablative cases on top of locative-cased nouns. For instance, *-il* + *-iruntu* = *-iliruntu* would function as ablative case marker for irrational nouns, as in (16). Similarly, *-iṭam* + *-iruntu* = *-iṭamiruntu* in turn functions as the marker for rational nouns.

(16)

நான்	கொழும்பிலிருந்து	வீட்டிற்கு	வந்தேன்
nāṉ	koḻump-il-iruntu	vīṭṭiṟku	vantēṉ
I.NOM	Colombo.LOC.ABL	home.DAT	come.PAST.1SR

'I came home from Colombo.'

4.3.2.7 Genitive case

உடைய *-uṭaiya*, அது *-atu*, இன் *-iṉ* are the markers used to mark genitive case in Tamil, which is available in the context of possession, and to show a thing's source or a characteristic/trait of something. There is no exact rule to say which marker needs to be used in what context (Schiffman, 2004). (17) shows how *-iṉ* is used to mark possessive. *-atu* is a confusing maker, which is also used to mark third-person, singular, and neuter in verbs and serves as a pronoun to mark the same. In addition, *-atu* functions differently in relative clause constructions. A genitive case example using *-atu* is shown in (18), as in this example, words can be shuffled when *-atu* is used to mark the genitive case. However, it is not possible with other genitive markers.

(17)

நான்	அவரின்	பிள்ளை
nāṉ	avariṉ	piḷḷai
I.NOM	he-HON.GEN	child.NOM

'I am his child.'

(18)

பள்ளிக்கூடம்	குமாரது
paḷḷikkūṭam	kumāratu
school.NOM	Kumar.GEN

'Kumar's school.'

Further, it is very common in Tamil that these genitive markers are dropped if it does not lead to any ambiguity. For instance, see (19), which is comparable to (17), where genitive case marker *-iṉ* is dropped, yet it shows the possessiveness.

(19)

நான்	அவர்	பிள்ளை
nāṉ	avar	piḷḷai
I.NOM	he-HON.NOM	child.NOM

'I am his child.'

4.3.2.8 Sociative case

ஒடு *-oṭu*, உடன் *-uṭaṉ* are the markers used to mark sociative cases in Tamil (Lehmann, 1993), see (20). When multiple entities function together as a subject, this conjunction will be reflected in subject-verb agreement, which is discussed in Section 4.3.8.

(20)

நான்	அப்பாவுடன்	போனேன்
nāṉ	appāvuṭaṉ	pōṉēṉ
I	father.SOC	go.PAST.1S

'I went with father.'

4.3.3 Agreement

Plurality agreement between a noun and its quantifier is obligatory for rational and irrational nouns. However, in the corpus I used, it is noted that this plurality agreement is not available for irrational nouns in some instances. I am not sure whether this is something that may have to do with regional differences. For instance, (21) was found in the corpus, although the alternative structure otherwise available in the Sri Lankan context is that in (22).

(21)

மூன்று	நாய்கள்	வந்தது
mūṉṟu	nāykaḷ	vantatu
three	dog.PL	come.PAST.3SN

'Three dogs came.'

(22)

மூன்று	நாய்கள்	வந்தன
mūṉṟu	nāykaḷ	vantaṉa
three	dog.PL	come.PAST.3PLN

'Three dogs came.'

A nominal subject and verbal predicate agree for rationality, gender, number, and person in Tamil. Tamil grammar textbooks, e.g. Nuhman (1999), claim that there are five gender values, as given below. Although this classification is referred to as gender, the markers do not express gender values alone. Rather it is gender + number values that are expressed.

1. āṇpāl - masculine singular
2. peṇpāl - feminine singular
3. palarpāl - rational plural
4. oṉṟaṉpāl - irrational singular
5. palaviṉpāl - irrational plural

As outlined in Chapter 3, I have used four gender values - masculine, feminine, epicene,[6] and neuter (Lehmann, 1998) without going into the additional detail and

[6]Epicene is used to mark singular-rational entities that are not classified with respect to masculine or feminine values.

complexity of GENDER and NUMBER markings being fused onto the same form.

When it comes to PERSON, there are three values, namely, first, second, and third, as in most other languages. However, Tamil has one more value that is used to mark a deictic reference between the second and the third person. Although special pronouns are used to address this medial deictic, this does not influence any syntactic functions, at least in the corpora I have handled. It has, for this reason, not been considered in my syntactic analysis. In addition to the three main person values, Lehmann (1993) has proposed the fourth person. According to him, this fourth person pronoun[7] in this section. — *tāṉ* — is always co-referential with the subject of the same clause or higher. In addition, there is no special agreement displayed between this fourth-person pronoun and the verb. The data I have used for the data-driven parsing has several instances of this fourth person — *tāṉ*. On the other hand, this *-tāṉ* can easily be confused with the *-tāṉ* clitic that is an emphatic marker in Tamil. Based on the data, I have treated *tāṉ* as clitic when it is written together with the headword, as in (23). Otherwise, *tāṉ* as a separately written word functions as a fourth person pronoun.

(23)

நான்	செய்தது	தவறுதான்
nāṉ	ceytatu	tavaṟutāṉ
I.NOM	do.PAST.REL	wrong.NOM.EMPH

'What I did was wrong.'

(24)

கண்ணன்	தான்	பரீட்சையில்	சித்தியடையமாட்டான்
kaṇṇaṉ	tāṉ	parīṭcaiyil	cittiyaṭaiya.māṭṭ.āṉ
Kannan.NOM	himself	exam.LOC	pass.will-not.3SMR

என்று	நினைத்தான்
eṉṟu	niṉaittāṉ
that	think.PAST.3SMR

'Kannan just thought he would not pass the exam.'

In Tamil, although there is a specific suffix with which to mark honorifics on verbal predicates, it is common to use the plural of suffix as a means to mark honorifics. Therefore, agreement for number may not always hold between the subject and the verbal predicate when this is the case, as shown in (26).

(25)

நான்	உன்னைப்	போகவிடமாட்டேன்
nāṉ	uṉṉaip	pōka.viṭa.māṭṭ.ēṉ
I	you.ACC.SANDHI-P	go.let.will-not.1S

'I will not let you go.'

[7]Although I believe that *tāṉ* is a reflexive pronoun, I have used the terminology of Lehmann (1993)

(26)

ஜனாதிபதி	கூட்டத்தில்	பேசினார்கள்
jan̲ātipati	kūṭṭattil	pēcin̲ārkaḷ
president.NOM	meeting.LOC	speak.PAST.3PLER

'The President spoke at the meeting.'

4.3.4 Negation

Negation can be realised morphologically using the *-aa* or *mattu* morphs, or by syntactically using the word *illai*. *-aa* is used to negative all types of verbal constructions, including finite and conditionals, as shown in (27). *-aa* can easily be confused with the question particle. *mattu* is specified as a future negative marker, as shown in (28) (Nuhman, 1999; Lehmann, 1993).

(27)

நீ	வராமலிருந்தால்	அப்பா	கோபிப்பார்
nī	varāmaliruntāl	appā	kōpippār
you	come-not-if	father.NOM	get-angry.FUT.3SER

'Dad will get angry if you don't come.'

(28)

நான்	உன்னைப்	போகவிடமாட்டேன்
nān̲	un̲n̲aip	pōka.viṭa.māṭṭ.ēn̲
I	you.ACC.SANDHI-P	go.let.will-not.1S

'I will not let you go.'

The word *illai* can function as a habitual (29), existential (30), and possessive (31) negator, as shown in the following examples from corpora:

(29)

நான்	அதிகாலையில்	எழும்புவது	இல்லை
nān̲	atikālaiyil	e̲lumpuvatu	illai
I	morning.LOC	get-up	no

'I do not wake up early.'

(30)

அவனுக்குப்	புத்தி	இல்லை
avan̲ukkup	putti	illai
he.DAT.SANDHI-P	wit.NOM	no

'He has no wit.'

(31)

எனக்கு	சந்தேகம்	இல்லை
en̲akku	cantēkam	illai
I.DAT	doubt.NOM	no

'I have no doubt.'

4.3.5 Postpositions in Tamil

Postpositions are sometimes referred to as particles (Arden, 1910), and are the equivalent to prepositions in English (Schiffman, 2004), except for their varied syntactic position, since postpositions are suffixed onto the words which they govern. However, analysing and listing all postpositions in Tamil is difficult since most of the postpositions are often nouns or verbs in their origin. Almost any verb in the language can be advanced to candidacy as a postposition (Schiffman, 2004). For instance, *pōṭu* 'put' and *koḷ* 'hold' can be used as postpositions with which to mark instrumental case (Schiffman, 2004) in modern usage. For example, (32) shows how *koḷ* 'hold' is used as a postposition to mark the instrumental case. However, if one can look deeper, it is clear that *koḷ* still functions as a verb and give the object case to stick. Otherwise, there are no ways to explain the case marking on the stick. Therefore, I have not handled all postpositions in Tamil as they require more linguistic exploration.

(32)

பொலீஸ்	தடியைக்கொண்டு	கூட்டத்தை	விரட்டியது
polīs	taṭiyaik-koṇṭu	kūṭṭattai	viraṭṭiyatu
police	stick.ACC-using	crowd.ACC	chase.PAST.3SN

'Police chased the crowd with stick.'

4.3.6 Verbs and their syntactic structures

Tamil verbs have been mostly analysed from a prescriptive perspective, and most of these studies are based on the very first Tamil grammar called *Tholkapiyam*[8] and a derived piece of work, the *naṉṉūl* (Thesikar, 1957), published in the 13th century CE. From the 18th century CE onwards, Western scholars have also contributed to the study of Tamil grammar. However, except for the attempt by Annamalai (2013), none of the scholars has clearly articulated the differences between complex predicates, serial verb constructions (Steever, 2005; Fedson, 1981), complex verbs (Agesthialingom, 1971), and compound verbs (Agesthialingom, 1971; Nuhman, 1999; Fedson, 1981; Paramasivam, 2011). This stands in contrast to the work done for other South Asian languages such as Urdu/Hindi (Butt, 1995; Butt and Lahiri, 2013). However, it is important to understand the differences across these potentially confusing categories for the development of grammar.

[8]The date of publication of this work is imprecise and uncertain. Scholars argue that it could be between the 5th century BCE and the 5th - century CE.

4.3.6.1 Complex Predicates

The study of complex predicates (CP) has received a great deal of attention in the linguistic literature and a number of distinct interpretations. I base this research on the definition proposed in Butt (1995), which views CPs as being formed when two or more predicational units enter into a relationship of co-predication. Each predicational unit adds arguments to a mono-clausal prediction; a similar definition or idea can also be found in Mohanan (1994) and Alsina et al. (1997). It is important to identify CPs to understand the differences with respect to other potentially confusing categories for the development of computational resources such as computational grammars (Butt and King, 2002), WordNet (Chakrabarti et al., 2007), and machine translation (Kaplan and Wedekind, 1993; Butt, 1994).

Complex predicates are very common in Tamil (Annamalai, 2013). For instance, verbs like வை (vay) 'place', விடு (vidu) 'let go', பார் (paar) 'see/look' may function both as main/full and light verbs. As light verbs, they mean 'cause', 'let' and 'try', respectively (Annamalai, 2013).

As noted in the existing literature (Annamalai, 2013; Steever, 2005; Lehmann, 1993; Rajendran, 2004), Tamil is well known for a diverse type of Verb-Verb (V-V) and Noun-Verb (N-V) constructions.

- verb+verb constructions involve a series of auxiliary or light verbs that are added periphrastically to the first verb, either in the form of a verbal participle or an infinitive. Muthuchchanmugan (2005) shows that up to four verbal units can follow the main verb (also referred to as the lexical headword) in Tamil. However, as he claims, whether all of these are auxiliaries is debatable. In V-V constructions, the terminal verbal unit is the final item in a sequence. The preceding verbal units can be in either an adverbial or infinitival form. The terminal verbal unit is the item that carries all the functional information, such as tense, person, number, and gender. The V-V sequences are used to express a range of semantic information. This includes cross-linguistically well-established categories such as the causative, passive, permissive, negation, aspectual information, and mood and modality, including obligation vs. possibility. The literature also describes definitive and conclusive meanings, the expression of irritation, carelessness, augmentation, prediction and intention (Paramasivam, 2011; Muthuchchanmugan, 2005). For instance, (33) shows construction with definite conclusive meaning, where the *muṭi* 'finish' is the head verb that gives the meaning of complete and the auxiliary verb *viṭu* 'leave' (in the example, *viṭu* has become *viṭ* due to the phonological transformation) marks the definite completion.

- noun+verb constructions where the noun function as the head and the verb as a light verb. (34) is an example for such a construction where *kaitu* 'arrest' is the

nominal head and *cey* 'do' is a light verb.

(33)

அமைச்சர்	கூட்டத்தை	முடித்துவிட்டார்
amaiccar	kūṭṭattai	muṭittu-viṭṭār
minister.NOM	meeting.ACC	conclude-definite.PAST.3SER

'The Minister concluded the meeting.'

(34)

பொலீஸ்	கள்வனைக்	கைது	செய்தது
polīs	kaḷvaṉaik	kaitu	ceytatu
police	thief.ACC.SANDHI-K	arrest.NOM	do.PAST.3SN

'The police arrested the thief.'

4.3.6.2 Light Verb Construction

Light Verbs (LV) differ from main/full verbs in their syntactic distribution and lexical semantics. While main verbs can stand alone and predicate independently, light verbs depend on the existence of another predicative element in the clause. LVs are light in the sense that they do not carry the meaning of the corresponding full verb, yet they still contain lexical-semantic information (Butt, 2010; Annamalai, 2013). Unlike auxiliaries, they are not fully functional elements. The light verb forms a syntactically monocausal unit with the main predicational element. Following Butt (2010), we analyse LVs as a separate syntactic category and differentiate them from both main verbs and auxiliaries in the language. Such structures are common in South Asian Languages (Butt and Lahiri, 2013), including Tamil (Annamalai, 2013).

Annamalai (2013) has analysed various V-V, Infinitive-V, N-V and Verbal Participle-V sequences that have been differentiated between Serial Verb Constructions (SVC) and Complex Predicates (CP). Example (35) illustrates a simple transitive verb வாங்கு (*vangu*) 'buy'. The same main verb used together with கொடு (*kodu*) 'give' in its light verb sense forms a CP in (36). The light verb 'give' contributes a beneficiary meaning to the predication and licenses the use of an additional beneficiary indirect object (OBJ-TH). The light verb as the terminal verbal unit carries functional information, which in this case has to do with tense, number and person.

(35)

நான்	காரை	வாங்கினேன்
naan	carai	vanginen
I.NOM	car.ACC	buy.PAST.1S

'I bought the car.'

(36)

நான்	அவனுக்குக்	காரை	வாங்கிக்கொடுத்தேன்
naan	avanukku-k	carai	vangikkoduththen
I.NOM	he.DAT-SAN	car.ACC	buy.VP.SAN.give.PAST.1S

'I bought him a car.'

Example in (37) shows an alternative version of (36), in which the two parts of the complex predication are realised together. The *Sandhi* [k] is also triggered on the main verb வாங்கு (vangu) 'buy' just as in the single word realisation in (36), thus further consolidating the monocausal function of the whole construct.

(37)

நான்	அவனுக்குக்	காரை	வாங்கிக்	கொடுத்தேன்
naan	avanukku-k	kar-ai	vangi-k	koduththen
I.NOM	he.NOM-SAN	car-ACC	buy.VP.SAN	give.PAST.1S

'I bought car for him.'

4.3.6.3 Causatives

Causative verbs indicate that one person or thing causes another to do something for another person. A causative in Tamil can be realised either morphologically or syntactically.

1. The morphological realisation of causation in Tamil

The causation in Tamil can be morphologically realised via three morphs: வி (vi), பி (pi), and ப்பி (ppi) occur before the tense maker in a verb (Steever, 2005). For instance, (38) shows how the causative marker வி (vi) is used to causative a verb. The choice of the causative morph depends on the last vowel of the verbal root and is thus phonologically conditioned (Nuhman, 1999; Paramasivam, 2011).

(38) வாங்குவித்தான் (vanguvittaan)

வாங்கு	-வி	-த்	-த்	-ஆன்
vangu	-vi	-t	-t	-aan
buy	-CAUS	-SAN	-PAST	-3SMR

'he made somebody buy (something)'

2. The syntactical realisation of causation in Tamil

The syntactical realisation of causation in Tamil can also be realised by adding one of the following verbs after the infinitive form of the main verb: செய் (sei) 'do', வை (vai) 'put', பண்ணு (pannu) 'do'. In this case, these verbs do not predicate as full verbs and have the character of light verbs, expressing the non-referential meaning 'make', as shown in (39).

(39)

அவனை	ஒரு	கார்	வாங்கச்	செய்தேன்
avanai	oru	car	vanga-c	seithen
he.ACC	a	car.NOM	buy.INF.SAN	make.PAST.3SM

'(I) made him buy a car.'

4.3.6.4 Passives in Tamil

Passive constructions in Tamil are realised via a V-V construction, and the verb படு (paṭu) 'be touched/be experienced/sleep' is used to passivise constructions, where *padu* functions as an auxiliary verb. Together with an infinitive form of the main verb, it gives the meaning of 'be subjected to' (Annamalai, 2013). For instance, (40) is a passive construction where paṭu is added to the infinitival form of *vāṅku* to passivise the construction. Further, agent '*rām*' become an instrumental oblique and object '*car*' has become nominative subject.

(40)

ராமால்	கார்	வாங்கப்பட்டது
rāmāl	kār	vāṅkappaṭṭatu
ram.INST	car.NOM	buy.INF.PASS.PAST.3SN

'A car was bought by Ram.'

(41)

ராம்	அவனுக்கு	ஒரு	கார்	வாங்கிக்	கொடுத்தான்
ram	avanukku	oru	car	vangki-k	koduththaan
ram.NOM	he.DAT	a	car.NOM	buy.VP.SAN	give.VP.PAST.3SM

'Ram bought a car for him.'

(42)

ராமால்	அவனுக்கு	ஒரு	கார்	வாங்கிக்	கொடுக்கப்பட்டது
ramaal	avanukku	oru	car	vangki-k	kodukkappaddathu
ram.INST	he.DAT	a	car.NOM	buy.VP.SAN	give.INF.SAN.PASS

'A car was bought for him by Ram.'

Further, V-V and N-V constructions can also be passivised. In such constructions, passiviser will be suffixed to the terminal auxiliary or light verb. For instance, consider (41) and its passive version in (42). As in causative CP constructions, passive constructions can also be written as two separate words, as in example (42), or like one token - வாங்கிக்கொடுக்கப்பட்டது (vāṅkikkoṭukkappaṭṭatu). However, in the corpus, I found that the passiviser verb *paṭu* is always written together as one token with the light verb as in (42) or the main verb.

4.3.6.5 Serial Verb Construction

Serial Verb Constructions (SVC) are common in Tamil. Unlike complex predicates, these constructions do not satisfy the constraint of co-predication and monoclausality, following the properties of SVCs outlined in Butt (1995), which differentiate SVC from other verbal predicates. (43) is an example of an SVC, where both *veddi* 'cut' and *vilttinaar* 'made-fall' share the same set of arguments and both of them are not bleached in their meaning. Anyway, SVCs need to be investigated in much more depth.

(43)

விவசாயி	மரத்தை	வெட்டி	வீழ்த்தினார்
vivasayi	marattai	veddi	vilttinaar
farmer	tree.NOM	cut.VPART	made-fall.PAST.3SER

'The farmer cut the tree down.'

4.3.6.6 Copula Construction

(44)

குமார்	வக்கீல்
kumār	vakkīl
Kumar.NOM	lawyer.NOM

'Kumar is a lawyer.'

(44) is an example of a copular construction with a nominal predicate marked with the nominative case. The copula verb *āku* '*lit.* to become' optionally occurs in nominal predicates. *āku* can be used to identify the predicate when the subject and the predicate are in the nominative case. For instance, we can conjoin *āku* only with *vakkīl* to form a meaningful sentence in (44). The negative copula verb *illai* appears obligatorily to express constitution negation as in Example (45).

(45)

குமார்	வக்கீல்	இல்லை
kumār	vakkīl	illai
Kumar.NOM	lawyer.NOM	not

'Kumar is not a lawyer.'

When the verb *illai* 'not' occurs as an existential negation, it is considered as the predicate as in example (46). There are instances where the nominal predicate is dative-case marked to express benefaction, as in example (47), and the copula is absent. In this example *āku* can only be suffixed to *kumārukku*, meaningfully; therefore, *kumārukku* is the predicate.

(46)

குமார்	வீட்டில்	இல்லை
kumār	vīṭṭ-il	illai
Kumar.NOM	home.LOC	not

'Kumar is not at home.'

(47)

இந்த	பரிசு	குமாருக்கு
inta	paricu	kumārukku
this	gift.NOM	Kumar.DAT

'This gift is for Kumar'

4.3.7 Clitics

Clitic is a word but cannot function independently because of its dependence on an adjoining word. Clitics are very common in Tamil. According to the Universal Dependencies guidelines, clitics should be separated from the main word to mark the corresponding syntactic functions in the dependency structure. This will be discussed later in Chapter 6, when discussing the data-driven parsing. In my analysis, I have only considered whether any clitics are attached. The only clitics found in the corpus data are: *-ā*, *-tāṉ*, and *-um*, which mark an interrogative structure, the emphatic, and inclusiveness, respectively. For instance, examples (49)- (42) shows how interrogative marker *-ā* is used along with the subject, the oblique noun, and the predicate to make different questions.

(48)

அவன்	கொழும்புக்குப்	போனான்
avaṉ	koḻumpukkup	pōṉāṉ
he.NOM	colombo.DAT.SANDHI-P	go.PAST.3SM

'He went to Colombo.'

(49)

அவனா	கொழும்புக்குப்	போனான்
avaṉā	koḻumpukkup	pōṉāṉ
he-who	colombo.DAT.SANDHI-P	go.PAST.3SM

'Is he who went to Colombo?'

(50)

அவன்	கொழும்புக்கா	போனான்
avaṉ	koḻumpukkup	pōṉāṉ
he.NOM	colombo.DAT.-where	go.PAST.3SM

'Is it to Colombo he went?'

(51)

அவன்	கொழும்புக்குப்	போனானா
avaṉ	koḻumpukkup	pōṉāṉā
he.NOM	colombo.DAT.SANDHI-P	go.PAST.3SM-did he

'Did he go to Colombo?'

4.3.8 Coordination

Conjunctions in Tamil can be marked with the use of the clitic *-um* or the token *mattum*. While the conjoined construction using *mattum* is straightforward, *-um* is ambiguous in many cases, as it has at least eight different functions in syntax such as in-completion, superiority, doubt, negation, completion, number, definiteness, and that which is to come Senavaraiyar (1938).[9] The conjunction works in the same way for nouns, adjectives, adverbs and verbs. Apart from these markers, a coordinate structure can also be marked using sociative cases on the coordinating noun phrases. When conjoined noun phrases occur as the subject, subject-verbal predicate agreement will follow the precedence shown below, where rationality (rat) always takes precedence.

$\text{NP}_{\text{1P+SG/PL+Rat}} + \text{NP}_{\text{1P+SG/PL+Rat}}\ (+\ \text{NP}_{\text{3P+SG/PL+Irrat}}) > \text{VC}_{\text{1P+PL+Rat}}$
$\text{NP}_{\text{1P+SG/PL+Rat}} + \text{NP}_{\text{2P+SG/PL+Rat}}\ (+\ \text{NP}_{\text{3P+SG/PL+Irrat}}) > \text{VC}_{\text{1P+PL+Rat}}$
$\text{NP}_{\text{2P+SG/PL+Rat}} + \text{NP}_{\text{2P+SG/PL+Rat}}\ (+\ \text{NP}_{\text{3P+SG/PL+Irrat}}) > \text{VC}_{\text{2P+PL+Rat}}$
$\text{NP}_{\text{2P+SG/PL+Rat}} + \text{NP}_{\text{3P+SG/PL+Rat}}\ (+\ \text{NP}_{\text{3P+SG/PL+Irrat}}) > \text{VC}_{\text{2P+PL+Rat}}$
$\text{NP}_{\text{3P+SG/PL+Rat}} + \text{NP}_{\text{3P+SG/PL+Rat}}\ (+\ \text{NP}_{\text{3P+SG/PL+Irrat}}) > \text{VC}_{\text{3P+PL+Rat}}$
$\text{NP}_{\text{3P+SG/PL+Irrat}} + \text{NP}_{\text{3P+SG/PL+Irrat}} > \text{VC}_{\text{3P+PL+Irrat}}$

Disjunction is marked with the use of the clitic *-oo* or the token அல்லது *allatu*. As in the case of coordination, this works in the same way irrespective of the disjunction of nouns, adjectives, adverbs, and verbs. Unlike the conjunction, it is unclear how the agreement between disjoined subjects and verbal predicates works. Since there were no such constructions in my corpora, I will not explore disjunction any further here.

4.3.9 Interrogatives

There are six types of interrogative structures in mentioned *naṉṉūl* (Thesikar, 1957). Their choice is based on the semantic nature of the question. Syntactically, all these questions can be constructed with the use of question particles or clitics *ō* and *ā*, or with the use of an in situ question pronoun, as shown in (52) and (53), respectively.

[9] http://www.tamilvu.org/courses/degree/a021/a0213/html/a021331.htm

(52)

நீ	தோசையா	சோறா	சாப்பிடுகிறாய்
nī	tōcaiyā	cō̲rā	cāppiṭukiṟāy
you	Dosa-*ā*	Rice.*ā*	eat.PAST.2S

'Do you eat dosa or rice?'

(53)

யார்	நாளைக்குக்	கொழும்புக்குப்	போகிறார்
yār	nāḷaikkuk	ko̲lumpukkup	pōkiṟār
who	tomorrow.DAT	Colombo.DAT	go.FUT.3SER

'Who is going to Colombo tomorrow?'

4.3.10 Complex clauses in Tamil

Complex sentences are constructed with the use of one or more subordinate clauses. The embedded clause or the subordinate clause is a subconstituent of the main clause. Mostly, the subordinate clause precede the main verb of the main clause. However, since Tamil displays free word order, subordinate clause can also follow the main verb of the main clause. This section briefly discusses the key types of complex sentence constructions I have encountered in the data I dealt with.

4.3.10.1 Non-finite clauses

Tamil has several types of non-finite clauses, including infinitives, adjectivals, adverbials, and conditional clauses. The infinitive clause can mark the semantic interpretations of the verb, including perception and cognitive. The infinitives are marked with *-a* and mostly analysed as open clausal complement structures.

(54)

நான்	குமாரை	கேட்க	வருகிறேன்
nā̲n	kumārai	kēṭk-a	varukiṟēn̲
I.NOM	Kumar.ACC	ask.INF	come.PRES.1S

'I come to ask Kumar.'

Adjectival clauses function like relative clauses, which will be discussed separately in Section 4.3.10.2. Adverbial clauses are constructed using the Verb+Adverbial-marker *-u* that modifies verbs, as shown in (55). These clauses are also used to construct the serial verb construction, as shown in 4.3.6.5.

(55)

மாணவன்	புத்தகத்தைத்	திறந்து	பார்த்தான்
māṇavan̲	puttakattait	tiṟantu	pārttān̲
student.NOM	book.ACC.SANDHI-T	open.VPART	see.PAST.3SM

'The student opened the book and looked.'

Conditional clauses are formed by Verb+Conditional-marker *-āl* that modify verbs, as

shown in (56). This should not be confused with the *-āl* marker used to mark various aspects, including the instrumental case and the agent in passive constructions.

(56)

கண்ணன்	வந்தால்	நான்	வர	மாட்டேன்
kaṇṇaṉ	vantāl	nāṉ	vara	māṭṭ.ēṉ
Kannan.NOM	come.CND	I.NOM	come.INF	will-not.1S

'I will not come if Kannan comes.'

4.3.10.2 Relative Clauses in Tamil

Relative pronouns do not introduce relative clauses (RCs) in Tamil as in English. RCs are formed by adding an *-a* morpheme to the verb (57), which Butt et al. (2020) refer to as a relatives. The relative marker is null in the future participle form with *-um*, as shown in (58).

(57) [angu **nin-ṛ-a**] paiyan-ai naan paar-t-en
there **stand**-PAST-REL boy-ACC I.NOM.1S see-PAST-1S
'I saw the boy who stood there.'

(58) [angu **nirk-um-∅**] paiyan-ai naan paar-pp-en
there **stand**-FUT-REL boy-ACC I.NOM.1S see-FUT-1S
'I will see the boy who will stand there.'

The head noun of the RC in (58) is 'boy', with the relative clause directly preceding the head. One also finds RCs without a head noun in predicative contexts, as in (59). In this case, the verb in the RC instead carries the pronominal form *-atu*, apart from the relativiser *-a*. This *-atu* is form-identical with the indefinite pronoun *athu*, hence the English paraphrasing with the use of 'one'.

(59) [angu nin-ṛ-a-**athu**] en thambi
there stand-PAST-REL-PRON.3SN my brother
'The **one** who stood there is my brother.'

(60) involves a full head noun 'boy' in the accusative as the matrix object, while (61) involves the substitution with the accusative pronoun *-avan* 'he' within the relative clause.

(60) [angu nin-ṛ-a] **paiyan-ai** naan paar-t-en
there stand-PAST-REL boy-ACC I.NOM see-PAST-1S
'I saw the boy who stood there.'

(61) [angu nin-ṛ-a-**van**-ai] naan paar-t-en
there stand-PAST-REL-PRON.3SM-ACC I.NOM see-PAST-1S
'I saw the one (he) stood there.'

We have done an initial study on relative clauses and reported it in Butt et al. (2020). However, a more in-depth study on relative clauses is required before handling relative clauses in my grammar-driven or data-driven parsers.

4.3.10.3 Complimentisers in Tamil

Tamil does not have complementisers of the *that*-type as in English. Rather, it uses a grammaticalised form of the verb *en* 'say', with the frozen past participle form *enṛu*, which has been analyzed as a type of quotative (Amritavalli, 2013; Balusu, 2020). (62) displays the ambiguity which results between a quotative use and a complementiser function. (63) illustrates a pure complementiser reading. Note that the matrix complementing verb can also take an accusative object as a co-referent for the complementiser clause (64). The pure complementiser form *enṛu* becomes *enṛa* by taking the relative marker *-a*. For further discussion on complementisers, see Butt et al. (2020).

(62) ravi [naan en nanban-ai santhi-tt-en] enṛu so-nn-an
Ravi.3SM.NOM [Pron.1S my friend-ACC meet-PAST-1S] QUOT say-PAST-3SM
'Ravi said that — "I met my friend".'
'Ravi said that I met my friend.'

(63) ravi [mazhai var-um enṛu] ninai-tt-aan
Ravi.3SM.NOM rain come-FUT.3SN COMP think-PAST-3SM
'Ravi thought that it will rain.'

(64) [avan pizhai sey-tt-aan enṛ-a] **unmaiy-ai** ram nirupi-tt-aan
he mistake do-PAST-3SM COMP-REL truth-ACC Ram.NOM prove-PAST-3SM
'Ram proved the truth that he made mistakes.'

4.4 Summary

In this section, I have discussed the background work that has been done on the basis of which parsing in Tamil could be made possible. I have also covered the Tamil syntax background required in order to understand the sentences found in the data collected from the Grade 1 Tamil textbook, ParGram sentences, grammar books, and short news texts. Among these different types of data sources, I have used the ParGram sentences and Grade-1 sentences to write rules for the grammar-driven parser and used the other data sources to train and evaluate the data-driven parser. Further to the given outline on Tamil syntax in this chapter, more details on complex predicates, complimentisers, and relative clauses in Tamil can be found in two research articles Sarveswaran and Butt (2019) and Butt et al. (2020) that we published in recent years. The following chapters will outline how I developed the grammar-driven and data-driven parsers.

5 GRAMMAR-DRIVEN PARSING

5.1 Lexical Functional Grammar (LFG)

Lexical-Functional Grammar (LFG) is a constraint-based non-transformational, generative formalism that uses a parallel architecture to model a grammar and represents different levels of linguistic information. LFG captures syntactic phenomena such as precedence, dominance, constituency, grammatical functions, predication, subcategorisation, and bounded and unbounded dependencies by using two representations, namely constituency structure (c-structure) and functional structure (f-structure) (Dalrymple, 2001; Falk, 2011).

The c-structure captures the phrase structure of the given sentences using a tree. The f-structure consists of functional information of a sentence/clause. This structure is called functional structure because, like in mathematical function, this structure also maps attributes to values (Dalrymple, 2001). An Attribute-Value-Matrix (AVM) structure is used to represent f-structure in LFG, as shown in (1). These AVMs structures can also be added inside another, as shown in (2).

As shown in (2) the f-structure also shows the grammatical function, where it has the syntactic head and the list of arguments it takes. In this example, it is just one argument — SUBJ. This argument structure can be changed based on the nature of syntactic construction during the parsing time using the restriction operator, which will be discussed in detail in Section 5.3.10.

(1)
$$\begin{bmatrix} \text{PRED} & \text{'lemma'} \\ \text{ATTRIBUTE1} & \text{value1} \end{bmatrix}$$

The f-structure can also contain sets to capture another f-structure consisting of attribute-value pairs and modifiers like adjectives and adverbs. (2) shows how two adverbs are captured using a set, which is given as a value of the attribute ADJUNCT.

(2) Sentence = 'He cried'

$$\begin{bmatrix} \text{PRED} & \text{'cry}\left\langle (\uparrow \text{SUBJ}) \right\rangle\text{'} \\ \text{TNS-ASP} & \begin{bmatrix} \text{TENSE} & \text{past} \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'he'} \\ \text{NTYPE} & \begin{bmatrix} \text{NSYN} & \text{pronoun} \end{bmatrix} \\ \text{PERS} & 3 \\ \text{GEND-SEM} & \text{male} \\ \text{CASE} & \text{nom} \end{bmatrix} \\ \text{VTYPE} & \text{main} \end{bmatrix}$$

(3) Sentence = 'He cried a lot yesterday'

$$\begin{bmatrix} \text{PRED} & \text{'cry}\left\langle (\uparrow \text{SUBJ}) \right\rangle\text{'} \\ \text{TNS-ASP} & \begin{bmatrix} \text{TENSE} & \text{past} \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'he'} \\ \text{NTYPE} & \begin{bmatrix} \text{NSYN} & \text{pronoun} \end{bmatrix} \\ \text{PERS} & 3 \\ \text{GEND-SEM} & \text{male} \\ \text{CASE} & \text{nom} \end{bmatrix} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{'a lot'} \\ \text{ADV-TYPE} & \text{vpadv} \end{bmatrix} \begin{bmatrix} \text{PRED} & \text{'yesterday'} \\ \text{ADV-TYPE} & \text{vpadv} \end{bmatrix} \right\} \\ \text{VTYPE} & \text{main} \end{bmatrix}$$

Although c-structure and f-structure use two completely different representations, namely a tree and AVM, to capture information, these two are linked together via projection functions (Butt et al., 1999; Dalrymple, 2001).

LFG is treated as a deep grammar formalism where what is captured using the f-structure is regarded as being mostly invariant among languages (This aspect has to do with the Universality principle of LFG). On the other hand, c-structures vary across languages and may display great variation in the syntactic structures they employ. This element of Universality makes it possible to develop parallel grammars. There has already been an effort to build a parallel LFG grammar for various languages (Butt et al., 1999), including South Asian languages such as Urdu and Hindi (Butt and King, 2002), as expanded upon below.

Beyond the c-structure and f-structure, other structures are used to capture several phenomena, such as anaphora, semantics, and the information expressed by the discourse. While LFG analyses have been extended beyond syntactic analyses, I have not addressed or made use of these structures in my account since the essence of this research is the development of a syntactic parser.

The LFG formalism will here be used to model Tamil structures. It has been successfully employed to model the grammar of morphologically and morphosyntactically complex languages, including South Asian languages such as Urdu and Hindi (Rehman and Kazi, 2017; Butt and King, 2002) and other complex and morphologically rich languages such as Murrinh-Patha (Seiss, 2012). Further, the Dravidian language scholar K.P. Mohanan has contributed to the development of LFG (Mohanan, 1997) as a contemporary of Prof. J. Bresnan, who co-invented LFG. Associated with the theoretical framework, an industry-grade grammar development environment called Xerox Linguistic Environment (XLE) is available to write grammars and parse text. LFG computational grammars have been created with the aid of XLE (Crouch et al., 2017; Butt et al., 1999), which also supports parsing and parse-bank generation (Sulger et al., 2013).

LFG further supports morphology integration, so we do not require developing a full-form lexicon. In a full-form lexicon, we would need to include each and every word in the grammar, which would be time-consuming and may make the grammar unnecessarily large for computation. In Tamil, a verb can take several hundred inflected forms (in common usage)(Annamalai et al., 2014). Even the morphological analyser I developed is able to handle 200+ forms for a single verbal lemma. Therefore, we can easily use a morphological analyser to handle all the required verb forms instead of writing every verb form. I have already developed a morphological analyser for this purpose, and I am now in the process of integrating it into the grammar, as I have described in this chapter.

5.1.1 ParGram project

The Parallel Grammar (ParGram) Project (Butt et al., 1999; Butt and King, 2002) aims to develop and implement large and wide coverage grammars for languages of different families. These parallel grammars are written collaboratively within the LFG's linguistic framework and an agreed set of grammatical features by the project group members. XLE is then used as a grammar development platform. In addition to putting effort into feature standardisation, the project also promotes similar analyses for similar phenomena across languages (Butt and King, 2002).

Since many languages are being parallelly modelled using LFG, machine translation has become easier, even if the languages involved have different features or are

typologically different. For instance, a machine translation system has been successfully developed between English and Murrinh-Patha using LFG (Seiss and Nordlinger, 2012). LFG has also been used in other language applications. For instance, it has been used to develop a fully-fledged, commercial-grade, question-answer system for the English language at the Palo Alto Research Center (Bobrow et al., 2007).

5.2 Methodology

To develop the Tamil grammar-driven parser, I first chose different corpora to identify different grammatical constructions. I then wrote phrase structure rules and lexical rules to parse each construction. This section describes each of the different construction I have been able to handle in my grammar as I then discuss the rationale behind the analysis and areas requiring more linguistic exploration.

5.2.1 Data

The ParGram project makes use of 100 sentences that cover important and most generally occurring syntactic structures in German[1] with which to develop parallel grammars in several languages. The sentences are translated into the respective languages before writing grammars. I translated these 100 sentences into Tamil and got those translations reviewed with a linguist's and two translators' support. These sentences do not cover all interesting syntactical constructions in Tamil. Therefore, apart from these 100 sentences, I have also used another 100 sentences from a school Tamil language textbook that is aimed at five-year-old children (Grade 1) by the Sri Lankan government on the basis of which to write my grammar rules. It was interesting that even the Grade 1 textbook offered a number of complex constructions, with some even requiring deep linguistic analyses.

5.2.2 Development Environment

I used the Xerox Linguistic Environment (XLE) (Crouch et al., 2017) to implement the parser, which is also used for the ParGram project. XLE is a re-implementation of the LFG Grammar Writers Workbench(Kaplan and Maxwell III, 1994) aimed at achieving more efficiency and production requirements. For instance, XLE can handle large online lexicons and can parse complex forty-word sentences in a reasonable amount of time (Butt et al., 1999).

XLE parses and generates sentences on the basis of the resources that are fed into it, namely phrase structure rules or grammar rules, lexicons, modules, namely tokenisers

[1] `https://www.ims.uni-stuttgart.de/en/research/projects/pargram/`

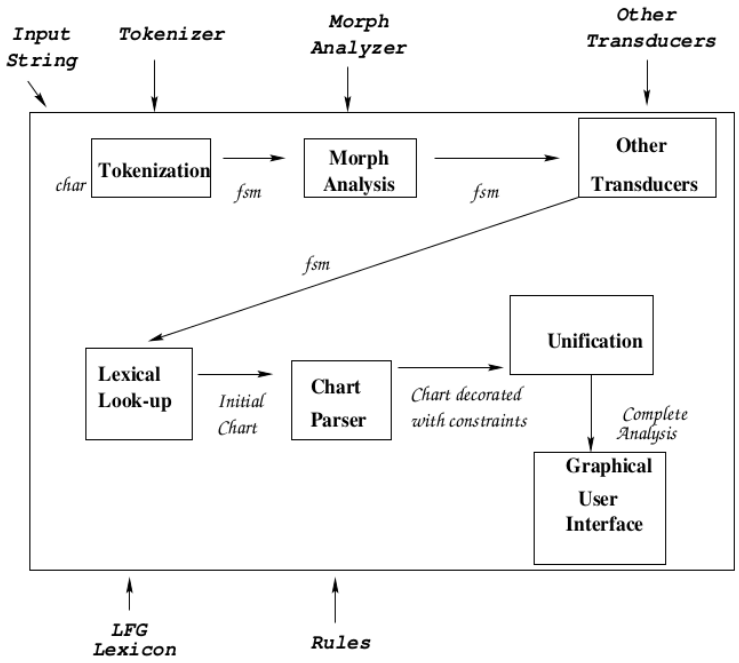

Figure 5.1: XLE architecture
Image source: (Butt et al., 1999)

for pre-processing and post-processing, and a Finite-State morphological analyser. In XLE, modules are developed using a device called a Finite-State machine. Besides these resources and modules, XLE uses a Chart parser with which to parse phrase-structure rules. Interactions between these modules and resources are shown in Figure 5.1, obtained from Butt et al. (1999).

Tokenisers are used in order to segment given texts into sentences. All the punctuations and multi-word syntactic tokens are separated within the sentence, while multi-word expressions are merged. This tokeniser is developed using a Finite-State machine. I have not developed a new tokeniser for Tamil. Instead, I have used the tokeniser that has been developed for the ParGarm project. Consequently, multi-words are currently not handled properly. The development of a good tokeniser would, in turn, require a corpus linguistic study, which is not within this study's scope.

The Finite-State morphological analyser is a morphological analyser that has been developed using the Xerox Finite-State Transducer (XFST) that supports analysis and generation. However, in the case of Tamil, the XFST did not help much due to its incompatibility issues for complex Unicode letters. I have instead made use of another FST tool called Foma (Hulden, 2009) on the recommendation of Lauri Karttunen. The development of the morphological analyser is discussed in detail in Chapter 3.

XLE has a graphical interface that is written using C and Tcl/Tk. However, the current version of the XLE interface has problems in rendering complex Unicode scripts like Tamil. Even after the software version and Tcl upgrades, vowel modifiers were

not correctly rendered. I spent a significant amount of time sorting this problem out with the help of various technicians who have worked with XLE. Eventually, I had to make use of the Infrastructure for the Exploration of Syntax and Semantics (INESS) platform[2] (Rosén et al., 2012) to write and test my grammar. INESS is a web interface that is written on top of the XLE framework, and this web interface renders Unicode scripts without any problem.

5.2.3 Annotation scheme

The ParGram project proposes an annotation scheme that documents morphosyntactic features to be used when writing a grammar that is being developed as part of a cross-lingual study.[3] I have primarily made use of this ParGram annotation scheme, but, I have then also extended some of these features and added a few new ones with which to capture the idiosyncratic morphosyntactic features found in Tamil, which are presented in Table 5.1.

5.2.4 Implementation

Phrase structure and lexical rules are written as plain text files and then fed to XLE for parsing and generation. Annotated phrase structure rules are used to capture the syntactic structure and primarily construct the c-structure. This also defines the constraints and mapping between the c-structure and the f-structure. Example (4) shows a simple phrase structure rule for English. This rule captures the fact that the head verb optionally consists of an object which is marked with accusative and zero or more post-positional phrases as a set.

(4)
$$\begin{aligned} \text{VP} \rightarrow \quad & \text{V: } \uparrow = \downarrow; \\ & (\text{NP: } \quad (\uparrow \text{OBJ}) = \downarrow \\ & \qquad\quad (\downarrow \text{CASE}) = \text{ACC}) \\ & \text{PP*:} \downarrow \in (\uparrow \text{ADJUNCT}). \end{aligned}$$

Lexical entries can be written in one or multiple text files, making lexicon management easier. I used separate lexicon files for nouns, verbs, and other word categories. (5) shows an example of a lexical entry for a given word form. This lexical rule for the lemma 'walk' denotes that the word can function as a verb or noun. The rule also marks the verb use of 'walk' as an intransitive verb that can take only a SUBJ. The '*' in the rule identifies the lexical entry as a complete one, and the parser is not required to do any lookup in the morphological analyser.

[2] http://clarino.uib.no/iness

[3] https://ling.sprachwiss.uni-konstanz.de/pages/xle/doc/PargramStarterGrammar/starternotes.html

(5)
$$\begin{array}{lll} \text{walk} & \text{V} * (\uparrow \text{PRED}) & = \text{`walk<}(\uparrow \text{SUBJ})\text{>'}; \\ & \text{N} * (\uparrow \text{PRED}) & = \text{`walk'} \end{array}$$

Table 5.1: The Tamil language specific morphosyntactic features

New features added		
Feature	Values	Description
RATIONALITY	+ / -	Rationality is an important morphosyntactic feature that is in Tamil used to classify entities which are rational or irrational. For instance, verbs agree with the subject on the basis of rationality, in addition to person, number, and gender. In subject-verb agreement, irrational entities take a specific number of gender suffixes, and rational entities take a distinct set. Therefore, while the nature of the gender suffix can decide whether a given subject is a rational entity or an irrational one, we wanted to mark rationality separately. This will be useful for cross-lingual analysis and helps reduce discourse analysis ambiguity. Apart from the subject-verb agreement, the rationality feature plays an important role when it comes to object marking. A rational object of a sentence is suffixed with the *ai* marker. On the other hand, this is not obligatory for irrational objects which are indefinite. However, an *ai* marker is obligatory if we want to mark the definiteness of irrational objects.
SANDHI	SANDHI-K SANDHI-P SANDHI-C SANDHI-T	Sandhi is another important morphosyntactic feature required for Tamil grammar.
Extended features values		
GENDER	EPI	Epicene is used to refer to the conflation of both masculine or feminine genders. It is also used as the gender in the case of honorific.

Templates in XLE are a useful resource that reduces a significant amount of manual work. I have taken the template file developed for ParGram[4] and extended it to accommodate the extended features that I have required explicitly for the Tamil grammar.

Finally, because of the rendering issues in XLE, I have made use of the INESS platform to test the rules by parsing text from the Grade 1 textbook and the ParGram project. The current INESS implementation makes use of the GIT platform[5] to feed rules and lexicons to the INESS platform. When I eventually got a proper parse, I downloaded the parsed data in Prolog format and added this Prolog file to the Tamil LFG treebank in the INESS platform. Since, at times, I got several parses for a given sentence, I was required to manually verify and choose the correct one to be added to the Tamil treebank.

The bigger challenge in implementing the grammar parser was the inadequate and confusing studies for the Tamil language. For this reason, I could not quite cover all of the ParGram and Grade 1 textbook sentences since these require more in-depth linguistic explorations which go beyond the remit of this work.

Section 5.3 below provides an outline of some of the constructions that have been handled in the grammar.

5.3 The LFG analysis of concepts and syntactic constructions

5.3.1 *Sandhi*

The CHECK feature, which was initially defined within ParGram to ensure morphosyntactic well-formedness, has been extended to include the relevant *Sandhi* information. The treatment of internal *Sandhi* is relatively straightforward in that words with an incorrect *Sandhi* pattern are not analysed and are thus identified as misspellings. However, external *Sandhi* had to be dealt with carefully. While we can find whether the given word has a *Sandhi* letter at the end or not, we cannot check whether the *Sandhi* used is correct prior to the syntactic analysis as it is contextual. I have therefore extended the CHECK feature to additionally check the well-formedness of the morphophonology by ensuring that the correct *Sandhi* letter is indeed available.

For example, for words with a *Sandhi* க் (k), I associate the f-structural attribute information (↑ CHECK _Sandhi-k) = +, which require the presence of this particular *Sandhi*. It is used to constrain the possible syntactic analyses in the grammar and to check whether the correct *Sandhi* has indeed been used in the syntactic context across

[4] https://ling.sprachwiss.uni-konstanz.de/pages/xle/doc/PargramStarterGrammar/starternotes.html

[5] https://git.app.uib.no/iness/pargram

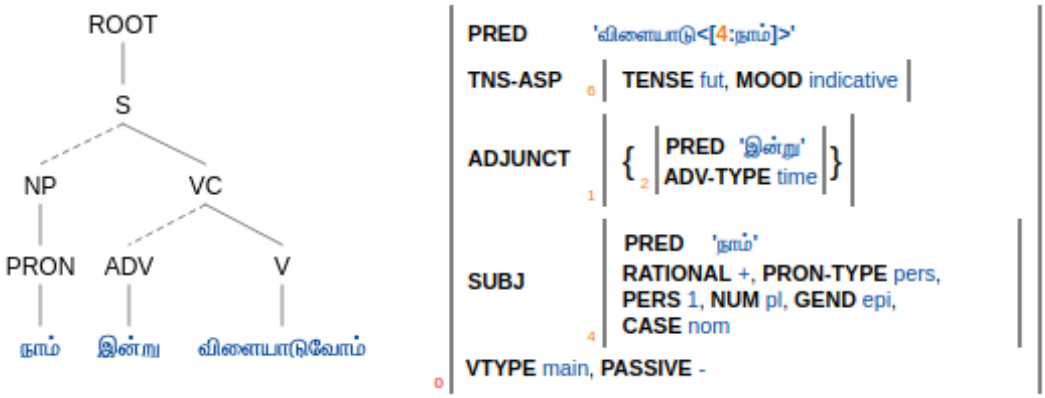

Figure 5.2: Analysis of an intransitive construction

two separate syntactic tokens. Instances of SANDHI-P, SANDHI-C, and SANDHI-T are treated similarly.

5.3.2 Intransitives

Example (6) provides a demonstration of an intransitive construction. Its analysis is illustrated in Figure 5.2. I have tried to include as many features as possible in the f-structure as proposed in the ParGram standards. The RATIONALITY, a feature we introduced, is used here to mark the subject of this construction as a rational entity.

(6)

நாம்	இன்று	விளையாடுவோம்
nām	in̲ru	viḷaiyāṭuvōm
We	today	play.FUT.1PLEPI

'We will play today'

5.3.3 Ditransitive construction

Ditransitive constructions have an indirect object, which is always marked with the dative case in Tamil.[6] (7) shows an example for a ditransitive construction found in ParGram, and Figure 5.3 and Figure 5.4 show the c-structure and f-structure of (7), respectively. This construction has the following important features that are captured in f-structure:

[6]As discussed in Chapter 4, in Tamil dative marker functions in different ways. Therefore, dative markers do not always mark indirect objects. If there are two NPs marked with the dative marker in construction, then the word order determines the indirect object; the first dative case marked noun phrase will serve as the subject, and the second one will serve as the indirect object. However, we have not handled the second pattern in our grammar yet as we did not find such construction in ParGram or in the Grade-1 textbook.

- Ditransitive verbs subcategorise for three arguments: subject, object, and indirect object or secondary object. In Tamil, the dative case of a noun phrase is used to indicate its status as an indirect object. In example (7), 'neighbour' bears the dative case marker, and the verb 'give' is a ditransitive verb. Therefore, 'neighbour' is marked as an indirect object. Indirect objects in ParGram specification are marked with a relation name OBJ-TH.

- 'tillage' is a nominal noun that modifies the following token 'machine', even though it is not overtly marked. This kind of nominal noun modification is common in Tamil.

- 'machine' bears an accusative case to mark the object of the construction. It also has CHECK is used to denote that it has a SANDHI-K marker; this is used to check whether the following token starts with SANDHI-K

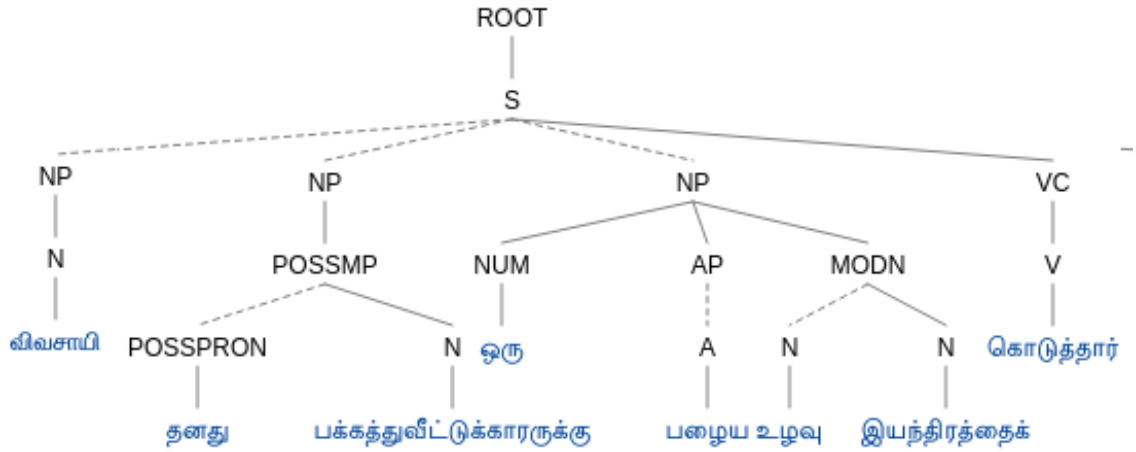

Figure 5.3: c-structure of (7)

(7)

விவசாயி	தனது	பக்கத்துவீட்டுக்காரருக்கு	ஒரு	பழைய
vivacāyi	taṉatu	pakkattuvīṭṭukkārarukku	oru	paḻaiya
farmer	his	neighbor.DAT	a	old

உழவு	இயந்திரத்தைக்	கொடுத்தார்
uḻavu	iyantirattaik	koṭuttār
tillage	machine.ACC.SANDHI-K	give.PAST.3SRE

'The farmer gave his neighbor an old tractor'

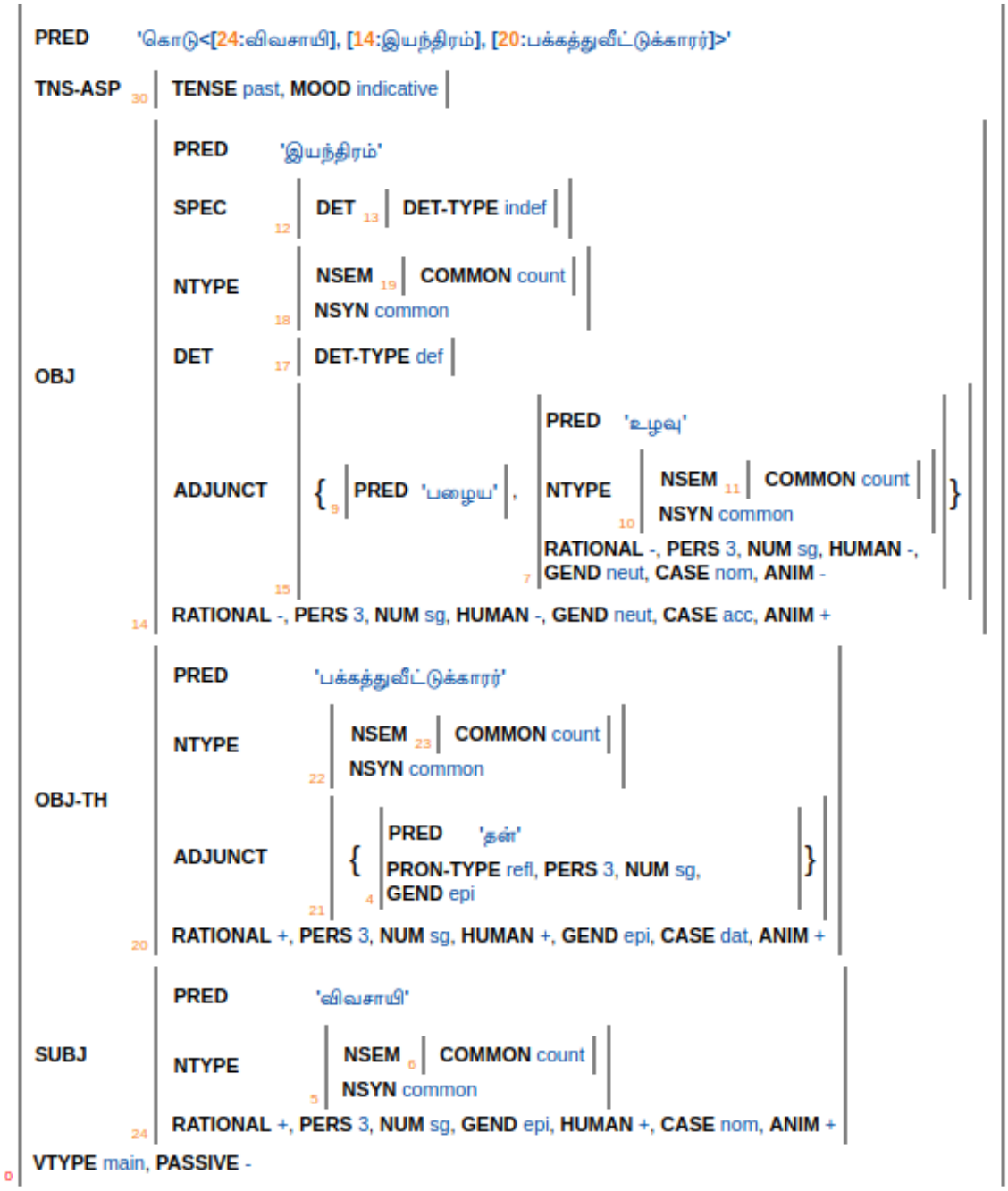

Figure 5.4: F-Structure of Example (7)

5.3.4 Imperatives

Example (8) is a negative imperative construction in Tamil. Tamil verb lemmata are imperatives. However, in this construction, an auxiliary is used to mark politeness on top of the lemma. The whole construction is negated morphologically using *ā*. I have made use of the feature ADDRESS as defined in the ParGram template to capture

politeness/respectfulness. The analysis produced by my grammar for (8) is shown in 5.5. Furthermore, since the object is an irrational entity and is not overtly marked for indefiniteness, the *ai* marker is used to mark definiteness in this construction, as discussed in Chapter 4.

(8)

பொத்தானை	அழுத்தவேண்டாம்
pottāṉai	aḻutta-vēṇṭ-ā-m
button.ACC	push-would-not.2RE

'Don't push the button.'

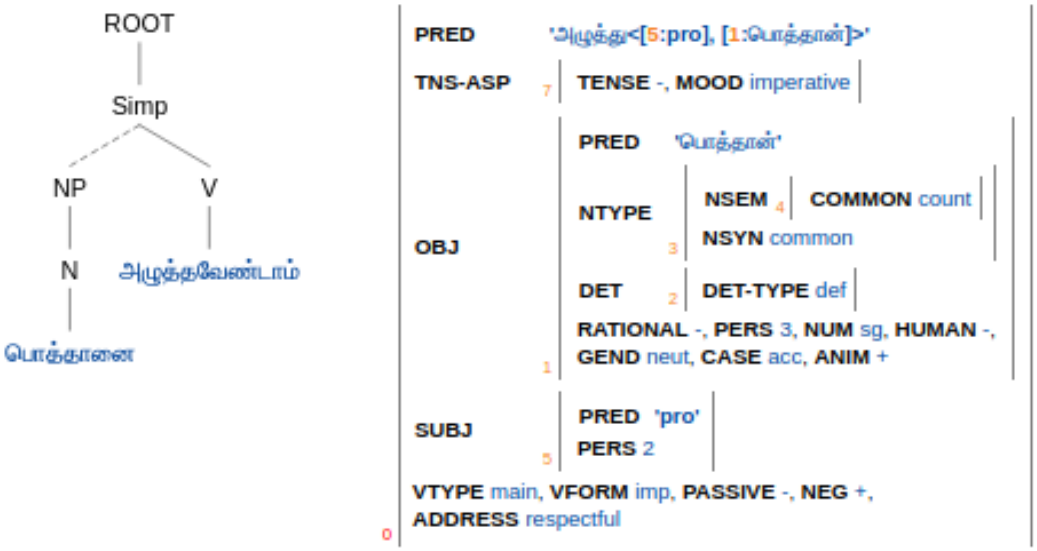

Figure 5.5: c-structure and f-structure analysis of (8)

5.3.5 Clausal argument structures

In addition to simple constructions that include arguments like SUBJ, OBJ, OBJ-TO, I have also handled clausal arguments structures which include either the finite-clause argument — COMP, or the non-finite-clause argument XCOMP.

COMP in Tamil generally behaves like *that*-clauses in English and is marked with என்று 'enṟu', which is a frozen complementiser in Tamil (Amritavalli, 2013). As discussed in Chapter 4, the embedded clause is a subconstituent of the main clause and can follow or precede the main verb of the main clause. Apart from this, there are also constructions where the complementiser is not marked overtly. Figures 5.6 and 5.7 provide the analysis for (9). Here the complementiser is overtly marked. In this example, *antap* is a demonstrative marker, and it is glossed as 'that'. Although the main verb follows the embedded clause in standard practice, the embedded clause can still be present in the middle of the main clause, as shown in (10)— this is a scrambled ver-

sion of (9). My phrase structure rules handle such word order as well. The c-structure analysis of (10) is given in Figure 5.8. In (9) and (10), *tāṉ* 'self' and the predicate in the embedded clause have the first person agreement, although the antecedent of the pronoun *tāṉ* 'self' is the third person entity *piḷḷai* 'child'.

உழவு	இயந்திரத்தை	தானே	இயக்கினேன்
uḻavu,	iyantirattai	tāṉē	iyakkiṉēṉ
tillage	machine.ACC	self.EMP	start.PAST.1SG

(9)

என்று	அந்தப்	பிள்ளை	நினைக்கிறது
enṟu	antap	piḷḷai	niṉaikkiṟatu
that	that.SANDHI-P	child.NOM	think.PRES-3SGN

lit: '[Tractor himself started] that that child thinks'
'The child thinks that he(self) started the tractor.'

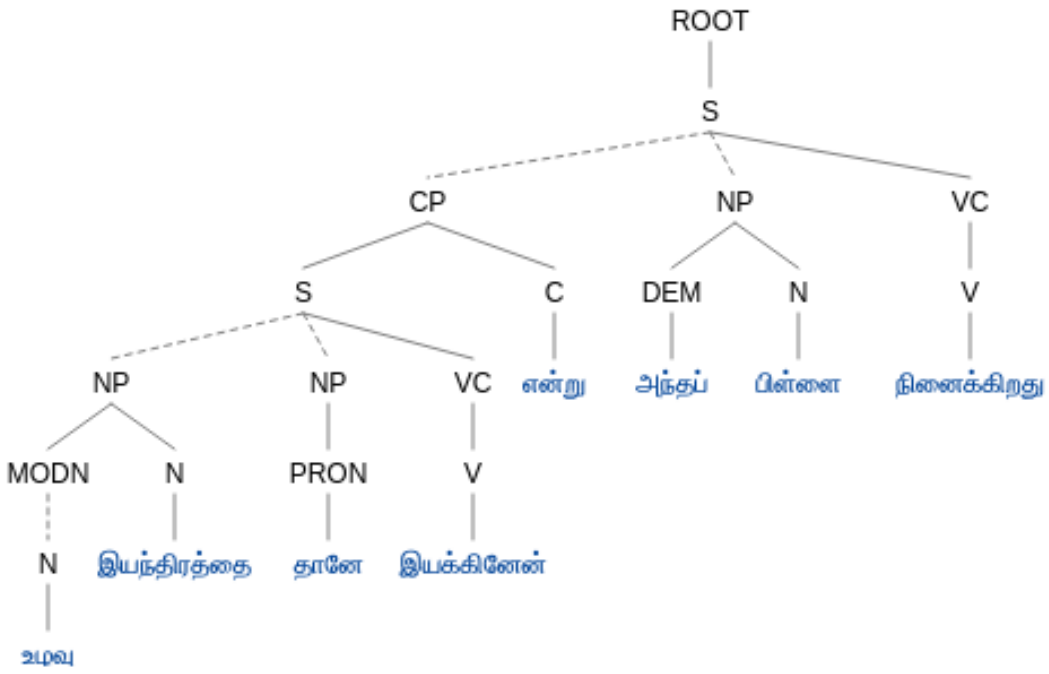

Figure 5.6: c-structure of a COMP analysis of (9)

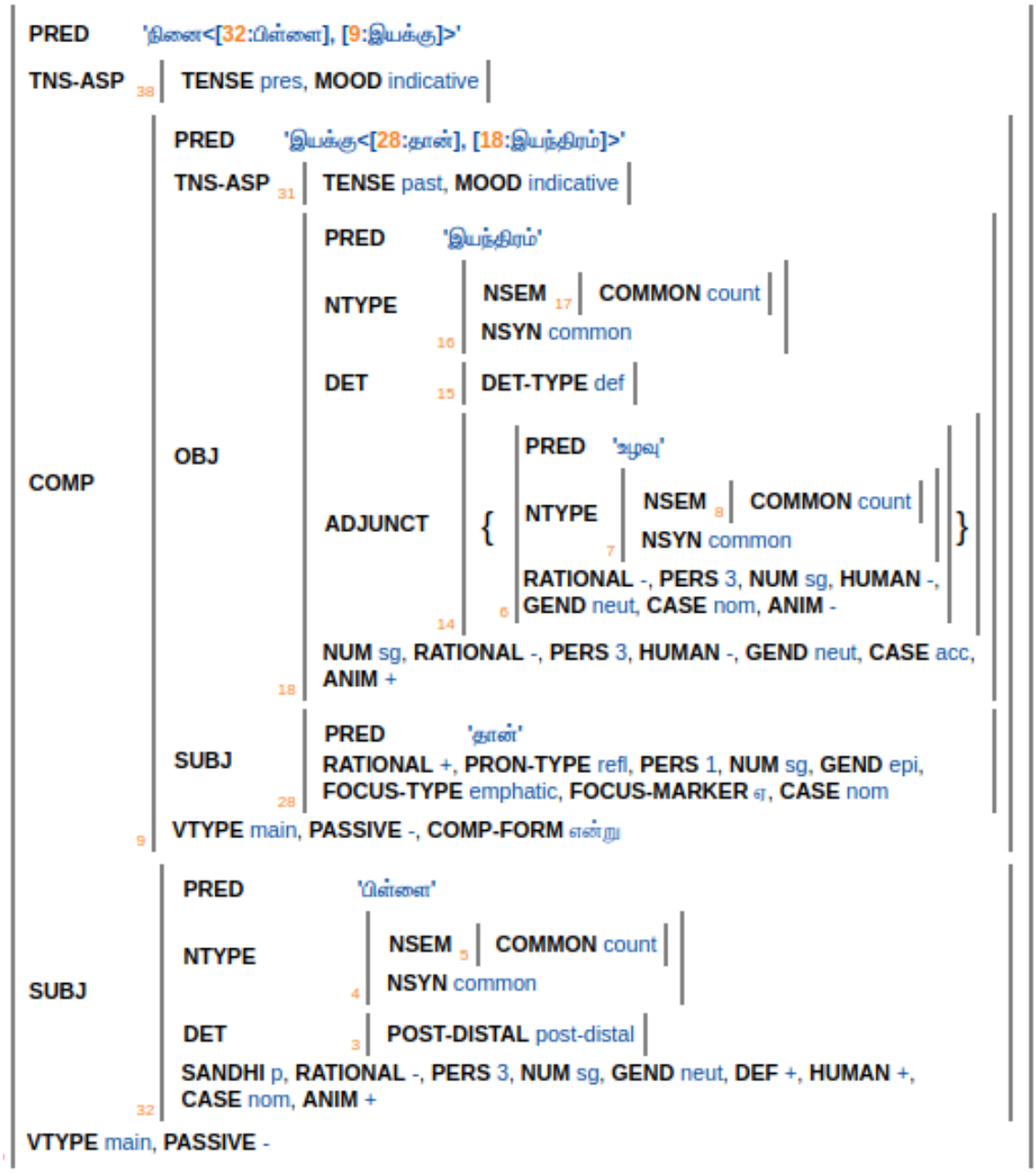

Figure 5.7: f-structure of (9)

	அந்தப்	பிள்ளை	உழவு	இயந்திரத்தை
	antap	piḷḷai	uḻavu,	iyantirattai
	that.SANDHI-P	child.NOM	tillage	machine.ACC

(10)

தானே	இயக்கினேன்	என்று	நினைக்கிறது
tāṉē	iyakkiṉēṉ	eṉṟu	niṉaikkiṟatu
self.EMP	start.PAST.1SG	that	think.PRES-3SGN

lit: 'that child [Tractor himself started] that thinks'
'The child thinks that he (self) started the tractor.'

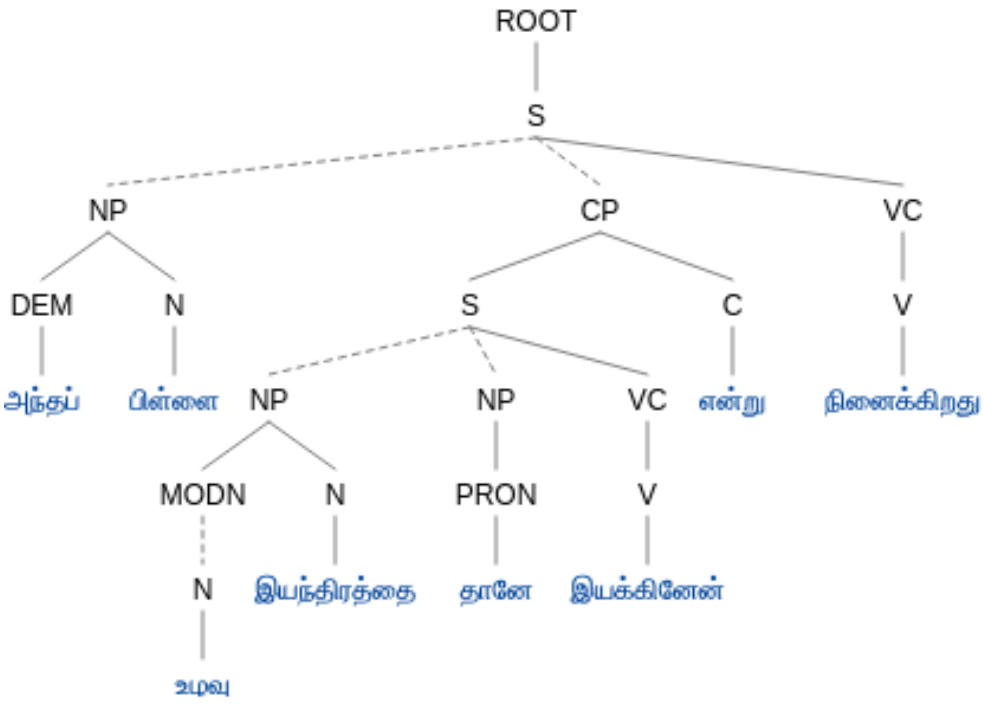

Figure 5.8: c-structure of (10)— a scrambled version of (9)

5.3.6 Oblique arguments in Tamil

Tamil has several types of OBLs, such as dative, locative, instrumental *etc.*. In this sub-section, I will show how datives are handled in my grammar as OBL arguments. Figure 5.9 shows how (11) is analysed.

(11)

பையன்	வீட்டுக்குப்	போனான்
paiyaṉ	vīṭṭukkup	pōṉāṉ
boy	home.DAT	go.PAST.3SMR

'The boy went home.'

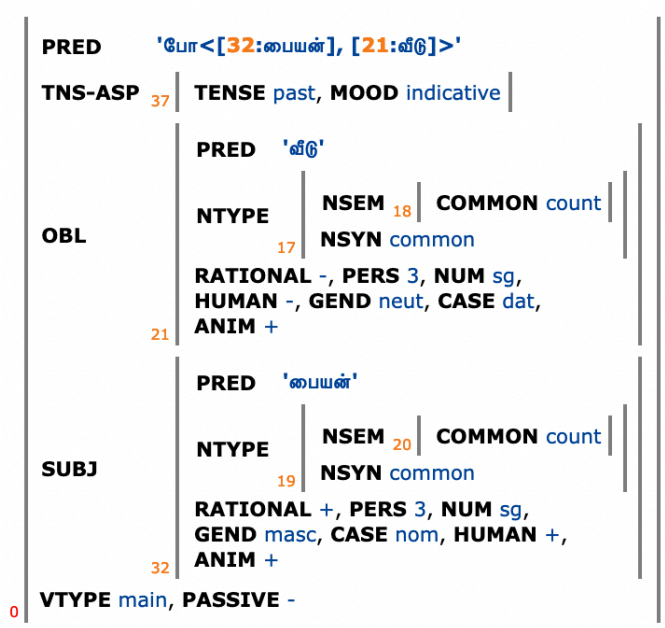

Figure 5.9: f-structure analysis of (11)

5.3.7 Interjection, Word-order, and Pro-drop

Interjections are commonly found in conversations in Tamil. Instead of just marking the token with the INTJ label, the token itself has been added to the f-structure analysis using the INTERJECTION-LABEL feature, as shown in Figure 5.10, as this information would be helpful to for semanticists to get the better meaning of the construction. The verb *iru* in Tamil has various functions. However, in example (12), it marks the existence of the theme. In the proposed analysis, it is analysed as a semantic head of the verbal predicate.[7]

(12)

அதோ ,	வாழைப்பழம்	இருக்கிறது
atō,	vāḻaippaḻam	iru-kkiṟ-atu
There,	banana	exist-PRES-3SGN

'There, Banana available (is existing)'

[7]One can argue that this is a gerundive noun and propose a PREDLINK analysis for this construction because *-atu* is a confusing suffix that is used to mark third-person, singular, neuter pronoun and used to show person-singular-neuter agreement between subject and verb.

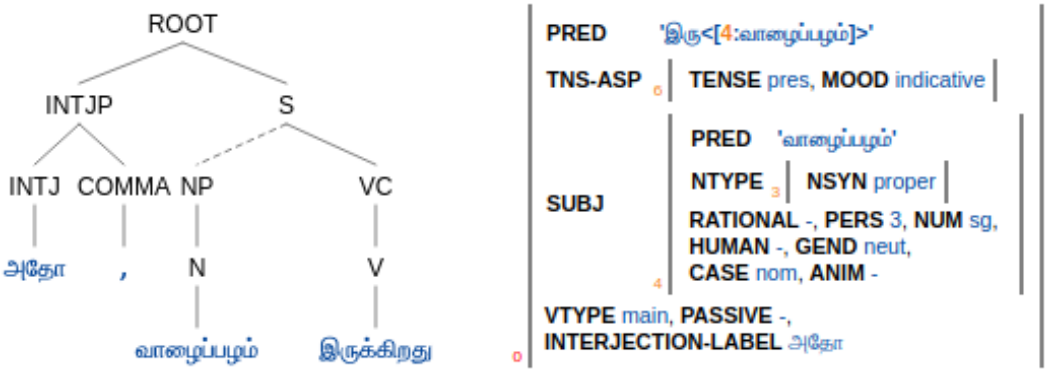

Figure 5.10: An example for an interjection

(13) involves yet another example of an interjection,[8] yet the word order involved is different. More importantly, this is also an example of pro-drop construction. Since the subject's properties are included in the suffix of the verbal predicate, pronominal subjects are not obligatory. Although the subject gets dropped, the sub-categorisation frame carries the 'pro' label, and then the properties of the subject are defined under SUBJ (King et al., 2005), as shown in Figure 5.11.

(13)

பலாப்பழம்	வாங்குவோம் ,	அம்மா
palāppaḻam	vāṅkuvōm	am'mā
Jackfruit	buy.FUT.1PLE,	mother

'Mother, we will buy Jackfruit'

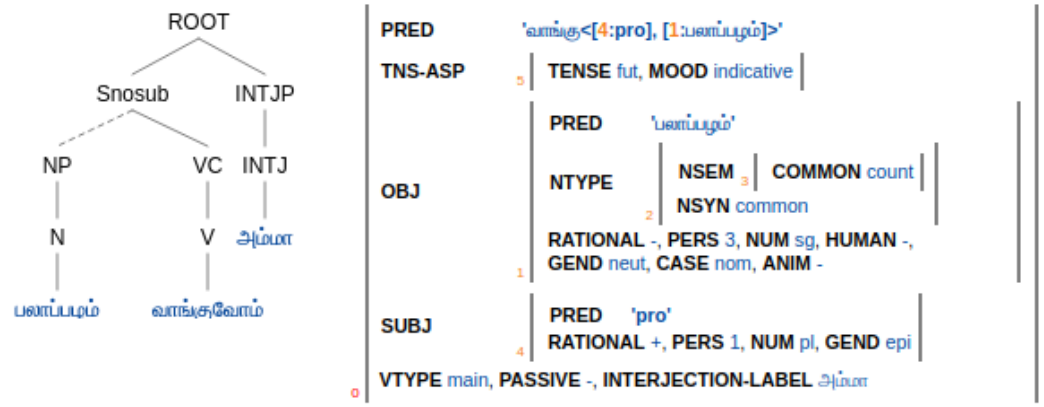

Figure 5.11: example for an interjection with a different word order, along with a representation of subject *pro*-drop

[8] One can argue that 'mother' is vocative, not interjection, in this construction. However, since the word does not have any particular grammatical relation to the main construction, I have analysed it as an interjection.

5.3.8 Copula and Null-Copula constructions

Tamil uses the copula *āku*, which forms N+N and N+A constructions. Null-copula constructions are also typical in Tamil. The grammar can handle both copula and null copula constructions. At times, we also found that the copula token is conjoined to the nominal predicate. In such cases, I handled the conjoined token as a predicate and analysed the structure accordingly.

Example (15) (a) shows how null-copula N+N constructions are analysed. Although there are two primary analyses for copula structures involving an XCOMP-PRED or a PREDLINK in association with the predicative material, since I am following the ParGram specification, I chose the PREDLINK analysis for copula constructions (Butt et al., 1999).

In PREDLINK analysis the subcategorisation frame of a copula verb (for instance, *āku* in Tamil) is represented as in (14). This representation model the fact that a particular property is predicated on the subject in a syntactically reasonable way. This analysis provides enough information for subsequent semantic analysis (Butt et al., 1999).

(14) (↑ PRED) = '*āku*<(↑ SUBJ)(↑ PREDLINK)>';

Figure 5.12 shows the f-structure analysis for (15) (a). As shown in (14), the copula verb *āku* is introduced as the head of the subcategorisation frame, even though it is a null head that is not present in the sentence. This analysis can be interpreted as there the subject of this construction Kumar of which a certain property, namely that of being a boy, is predicated, as indicated by the grammatical function PREDLINK.

Example (15) (b) shows a parallel structure to (15) (a), but (15) (b) involves a copula in the construction. From the analyses given in Figure 5.13 and Figure 5.12, we can see that while the c-structures vary between both structures, the f-structure analyses are the same.[9]

(15) (a).

குமார்	பையன்
Kumār	paiyaṉ
Kumar	boy.NOM

'Kumar is a boy.'

[9]In (15) (b) and in the analysis shown in Figure 5.13, சிவப்பு *civappu* 'red' is marked as a noun. In Tamil, red behaves as a noun and takes all the cases. Therefore, it is considered a noun in this analysis.

(b).

உழவு	இயந்திரம்	சிவப்பு	ஆகும்
uḻavu	iyantiram	civappu	ākum
tillage	machine	red	be

'The tractor is red.'

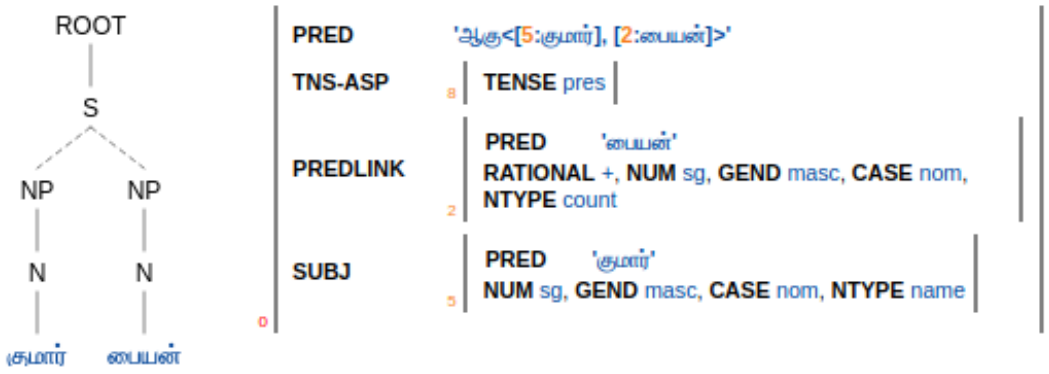

Figure 5.12: Analysis of the null-copula construction given in Example (15) (a)

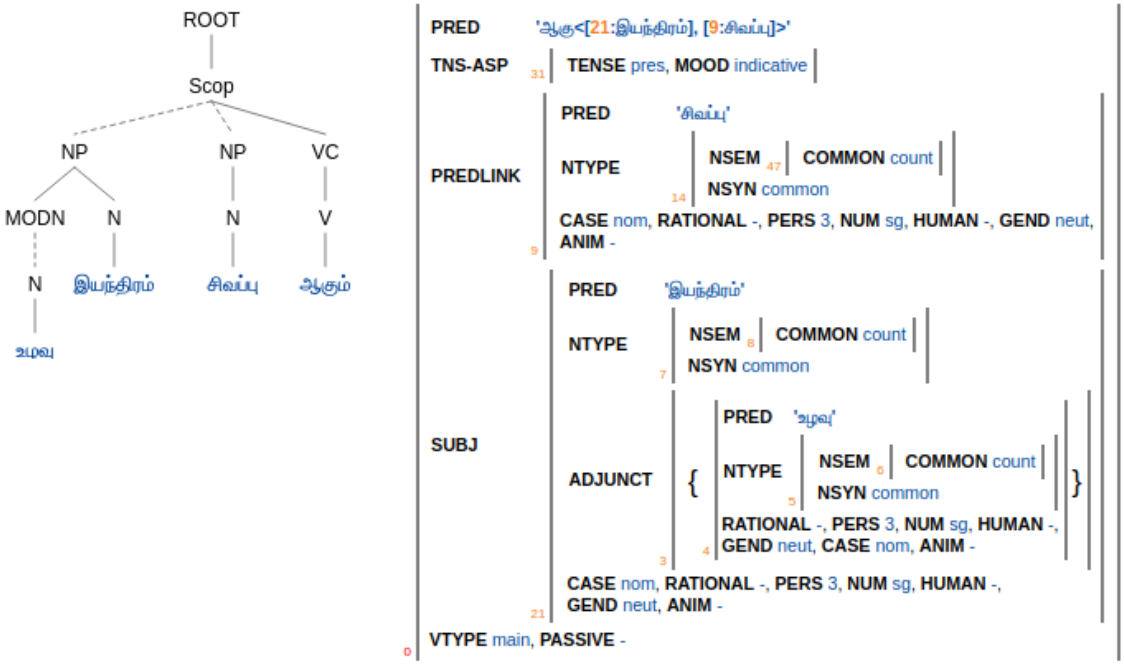

Figure 5.13: Analysis of the copula construction given in Example (15) (b)

As shown in (16), Tamil also has N+A construction. In this case, the copula verb is conjoined with the adjective. Similarly, a copula verb can also be conjoined with a noun. In this sort of construction, I have considered that conjoined token as the predicate, as shown in the analysis in Figure 5.14.

(16)

கோயில்	கோபுரம்	உயரமானது
kōyil	kōpuram	uyaramāṉatu
temple	tower.NOM	tall-be

'The temple tower is tall.'

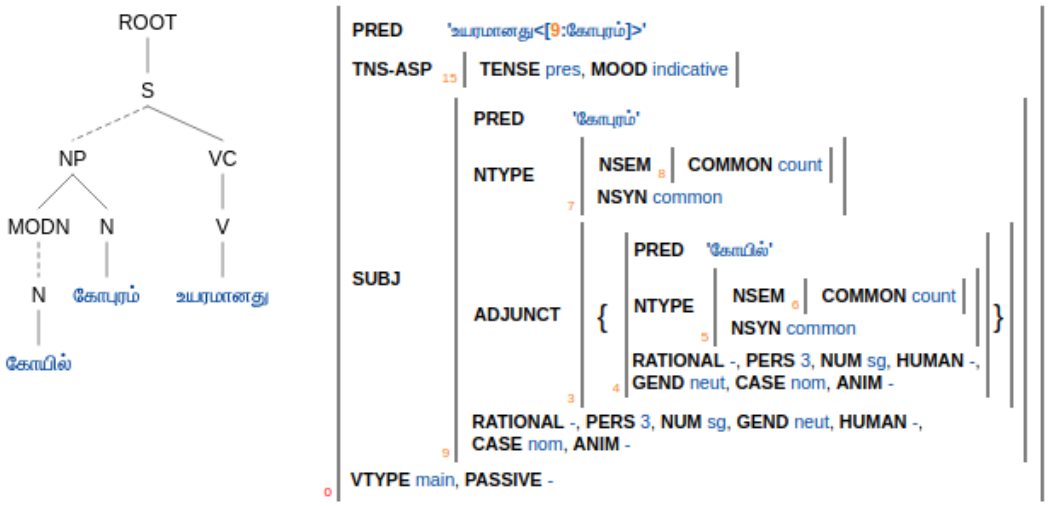

Figure 5.14: Analysis of the N+A copula construction given in Example (16)

5.3.9 Dative Subjects

As discussed in Chapter 4, dative subjects are common in Tamil. The corpora I used also has such constructions; one such instance is (17). Figure 5.15 shows how this is analysed in my grammar. As in the analysis, dative subject trigger 3SN agreement on the verb, even if the dative subject is a rational entity or has different PERSON or GENDER feature values as in (17).

(17)

ஆம்,	எனக்குக்	கேட்கிறது
ām,	eṉakkuk	kēṭkiṟatu
yes,	I.DAT	hear.PRES.3SN

'Yes, I hear (it)'

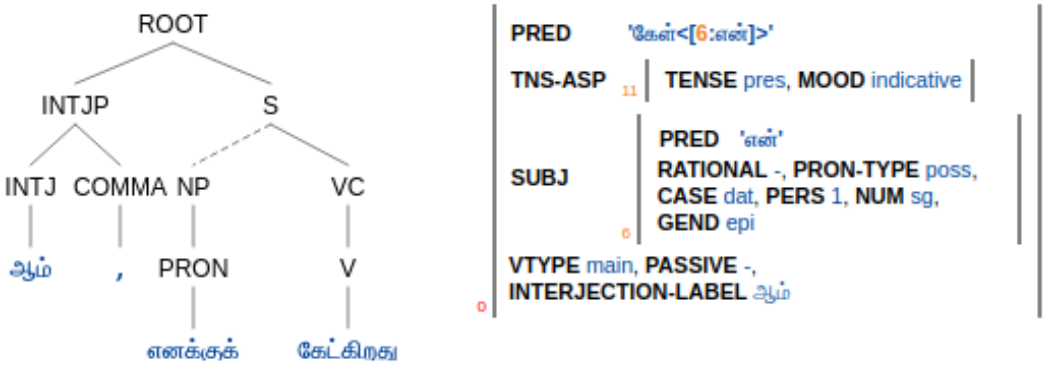

Figure 5.15: Analysis of the dative subject construction in (17)

5.3.10 Complex Predicates and handling of multiple predicational heads

In LFG, semantic heads of complex predicates correspond to a single PRED at the level of f-structure. C-structure does not determine complex predicate status, and the elements contributing to a complex predication can be either morphological or syntactic (Butt, 2010). However, regardless of whether the complex predication is morphological or syntactic, the composition of the arguments of both of the predicational units works according to the same principles (Alsina, 1996).

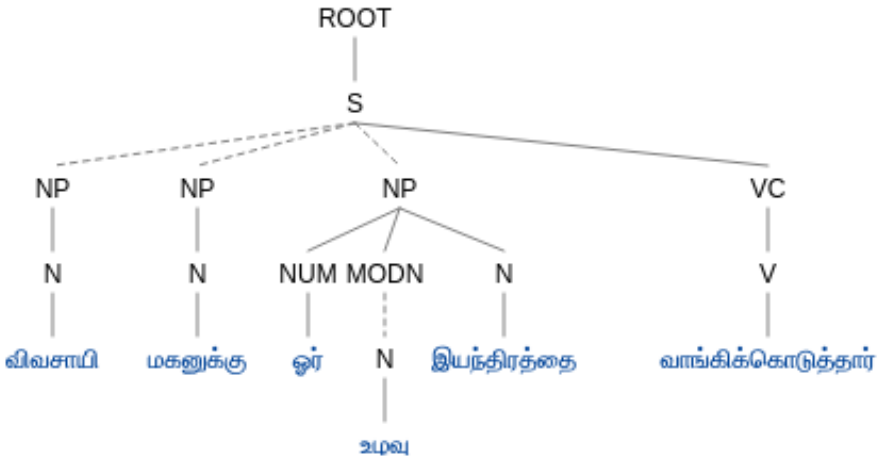

Figure 5.16: C-structure analysis of (18), where semantic heads of a complex predicate are written together

When a complex predicate is written as one token as in வாங்கிக்கொடுத்தார் *vāṅkikkoṭuttār* 'He bought for someone', which is composed on buy+give, we provide a subcategorisation frame as part of the lexical rule and handle it as a regular lexical item where வாங்கிக்கொடு *vāṅkikkoṭu* 'buy+give' together is considered as the complex root of the complex predicate *vāṅkikkoṭuttār*. In this case, all three arguments, namely subject,

object, and indirect object, are categorised by this complex root. For the example in (18) Figures 5.16 and 5.17 provide the respective c-structure and f-structure analyses. Here *vāṅkikkoṭuttār* 'he bought for someone' is handled via a lexical rule.

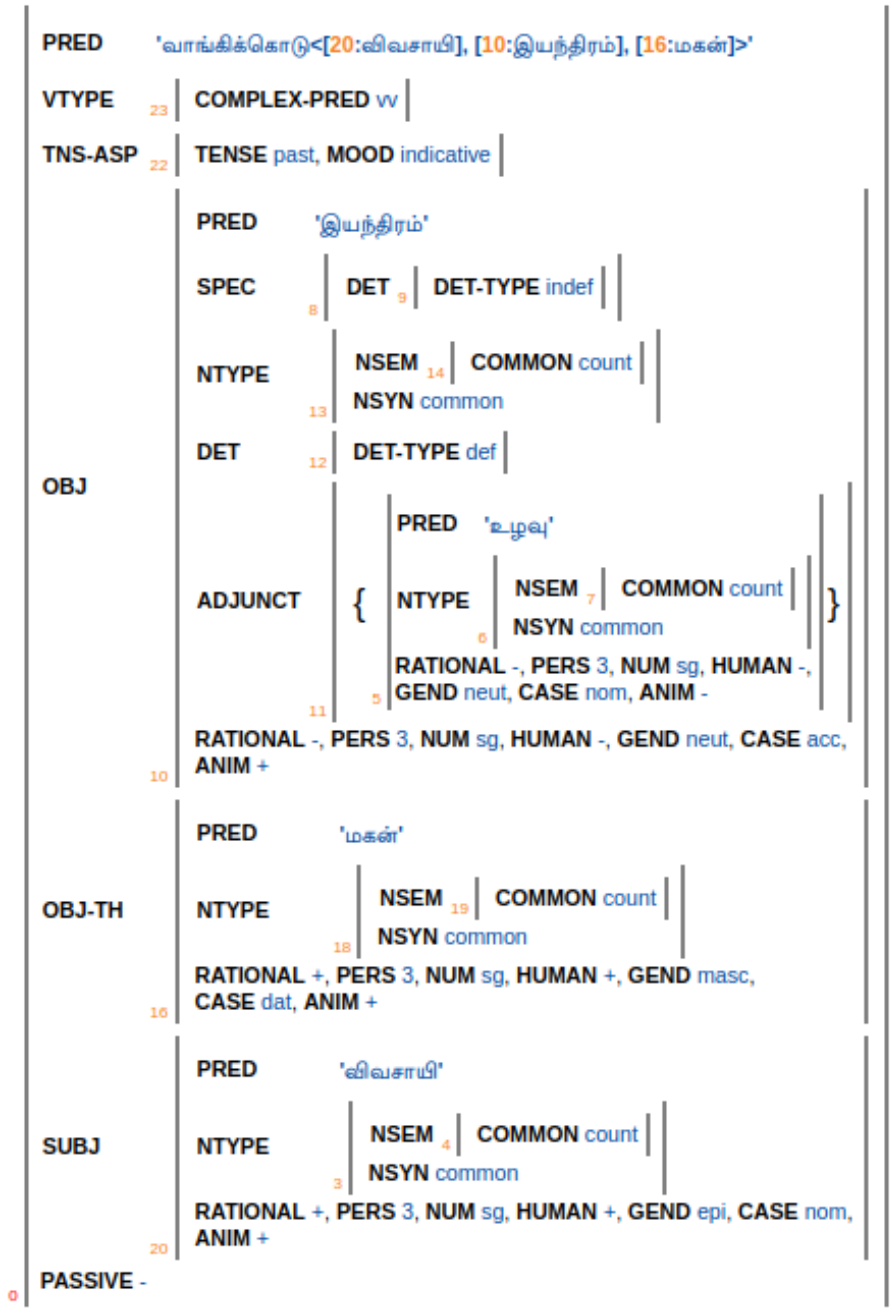

Figure 5.17: F-structure analysis of (18), where the semantic heads of the complex predicate are written together

விவசாயி	மகனுக்கு	ஓர்
vivacāyi	makaṉukku	ōr
farmer	son.DAT	a

(18)

உழவு	இயந்திரத்தை	வாங்கிக்கொடுத்தார்
uḻavu	iyantirattai	vāṅkikkoṭuttār
tillage	machine.ACC	buy-give.PAST.3SE

'The farmer bought his son a tractor.'

When a complex predicate is written as two tokens, the two predicational units need to be composed. For instance, when வாங்கிக்கொடுத்தார் *vāṅkikkoṭuttār* 'He bought for someone' written as two tokens, வாங்கிக் + கொடுத்தார் *vāṅkik + koṭuttār* 'buy + give', we need to find a way to do predicate composition. Because, as shown in (19) and (20), subject and object are subcategorised by the main verb *vāṅki* 'buy', and the indirect object is subcategorised by the light verb *koṭuttār* 'give'. However, in the analysis, we need to compose them.

(19) வாங்கிக் V * (↑ PRED)='வாங்கு<(↑SUBJ)(↑OBJ)>' .

(20) கொடுத்தார் Vlight * (↑ PRED)='கொடு<(↑OBJ-TH)>' .

The Restriction Operator (Kaplan and Wedekind, 1993) allows for the manipulation of f-structural information. For instance, Attribute-feature values may be "restricted" out as shown in the f-structures in (21) and (22), where the CASE feature has been restricted out of (21) via an application of the Restriction Operator '/': (↑/CASE). This type of restricting out might become necessary if a value for CASE had already been assigned by one part of the grammar but needed to be changed by another. This restrictive nature of Restriction Operator can also be extended to do predicate composition.

(21)
$$\begin{bmatrix} \text{PRED} & \text{'Ram'} \\ \text{PERS} & 3 \\ \text{NUM} & \text{SG} \\ \text{CASE} & \text{NOM} \end{bmatrix}$$

(22)
$$\begin{bmatrix} \text{PRED} & \text{'Ram'} \\ \text{PERS} & 3 \\ \text{NUM} & \text{SG} \end{bmatrix}$$

I follow studies that use the Restriction Operator in the context of complex predicate formation (Butt et al., 2003; Butt and King, 2003; Butt et al., 2008) and passivisation

(Wedekind and Orsnes, 2003). The Restriction Operator has also been used to implement the grammar when the light verb subcategorises for regular arguments as well as a variable %PRED2. The main verb's subcategorisation frame substitutes this variable as part of the complex predicate composition. For instance, in our grammar, I have a lexical entry for a light verb with its functional features கொடுத்தேன் *koṭuttēṉ* 'I gave', as shown in (23). I have then used the template in (24), obtained from Dalrymple et al. (2004) in the grammar rules for the light verb to compose arguments.

(23) கொடு Vlight * (↑ PRED)='கொடு<(↑OBJ-TH) %PRED2>' .

(24)

VV-ANNOTATION =
(↓ CHECK _RESTRICTED) = +
(↑ PRED ARG2) = (↓ PRED)
↓\PRED\SUBJ\CHECK\ = ↑\PRED\SUBJ\CHECK\
OBJ-TH\OBJ\PASSIVE OBJ-TH\OBJ\PASSIVE
(↑ OBJ) = (↓ OBJ)
(↑ SUBJ) = (↓ SUBJ)
(↑ VTYPE COMPLEX-PRED) = vv.

5.3.11 Passives

Passive constructions in Tamil are realised via a V-V construction and the verb படு *padu* 'be touched/be experienced/sleep' is used to passivise constructions, where *padu* is an auxiliary verb. Together with an infinitive form of the main word, it gives the meaning of 'be subjected to' (Annamalai, 2013).

For instance, consider (25) and its passive version in (26). As per the standard LFG lexical rule for passivisation, the original OBJ becomes the nominative SUBJ and the original SUBJ is realised as an instrumental adjunct (OBL-INST). Passives can be written as one word; for our example, in (26) this is வாங்கிக்கொடுக்கப்பட்டது *vāṅkikkoṭukkappaṭṭatu* 'was bought'. The passiviser *padu* can also be written as separate words as shown in (27). However, in the corpus, the passive auxiliary is always written together with it, as in (26).

(25)

ராம்	அவனுக்கு	ஒரு	கார்	வாங்கிக்	கொடுத்தான்
ram	avanukku	oru	car	vangki-k	koduththaan
ram.NOM	he.DAT	a	car.NOM	buy.VP-SAN	give.VP.PAST.3SG

'Ram bought a car for him.'

(26)

ராமால்	அவனுக்கு	ஒரு	கார்	வாங்கிக்	கொடுக்கப்பட்டது
ramaal	avanukku	oru	car	vangki-k	kodukkappaddathu
ram.INST	he.DAT	a	car.NOM	buy.VP-SAN	give.INF.SAN.PASS

'A car was bought for him by Ram.'

(27)

கொடுக்கப்	பட்டது
kodukkap	paddatu
give.INF.SAN	do.PAST.3SG

'was given'

(28) shows a passive construction found in our corpora, and Figure 5.18 gives the illustration of c-structure and f-structure analyses.

(28)

மரம்	நேற்று	வெட்டப்பட்டது
maram	nēr̲ru	veṭṭappaṭṭatu
tree.NOM	yesterday	cut.PASSIVE

'The tree was cut yesterday.'

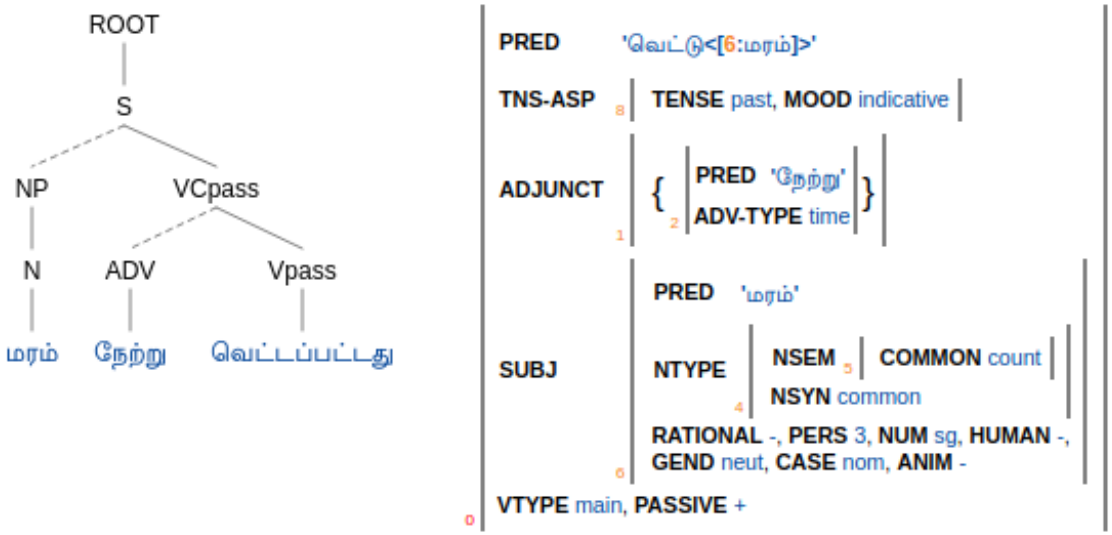

Figure 5.18: c-structure and f-structure analyses of a simple passive construction given in (28)

If the predicational units of a verb are written together, they can be dealt with the morphological analyser as shown in (29).[10] The surface form of the word is associated with tags shown in (29) via a series of rules. The analysis provides the information that this was a passive verb in the past and combined with a 'give' light verb.

[10]Effects of assimilation and *Sandhi* are shown within parentheses.

(29)

வாங்கிக்கொடுக்கப்பட்டது
vangikkodukkappattathu
vang-i-(k)kodu-(k)k(ap)-pattatu
vangu +verb +vp +give +past +pass

The stem (*vangu*) and the tag for the light verb 'give' are straightforwardly associated with the subcategorisation information via the stem lexicon contained in the grammar, as shown in (23) and (30).

(30) வாங்கு V * (↑ PRED)='வாங்கு<(↑ SUBJ)(↑ OBJ)>' .

Most of the attendant morphological tags are also straightforwardly associated with the corresponding f-structure information, e.g., (↑ TNS-ASP) = PAST for the '+past' tag (Kaplan et al., 2004). However, treating the passive is interesting.

Classically, the passive is treated via lexical rules (Bresnan, 1982) in the ParGram grammars. These lexical rules are coded as part of the lexical entries, allowing for either an active or passive verb version. The passivised version of vangu 'buy' in (31), for example, would result in the subcategorisation frame (↑ PRED) = 'vangu<(↑ SUBJ)>' due to the application of the passive lexical rule.

(31)
vangu	(↑ PRED) = 'vangu<(↑ SUBJ) (↑ OBJ)>'
+past	(↑ TNS-ASP TENSE) = PAST
+3sg	(↑ PERS) = 3
	(↑ NUM) = sg

Passivisation can also be applied to a composed complex subcategorisation frame. The composed subcategorisation frame of the complex predicate வாங்கிக்கொடுத்தான் *vāṅkikkoṭuttāṉ* '(he) bought for someone' is 'give-buy <(↑ OBJth) (↑ SUBJ) (↑ OBJ)>', for instance. When it is passivised, the resulting subcategorisation frame would be 'give-buy <(↑ OBJth) (↑ SUBJ)>'. Passivisation via lexical rules is straightforwardly implementable when the parts of the verbal sequence are contained within one lexical item. In this case, the lexical rule can be applied to the whole composed subcategorisation frame that is coming out of the lexicon.

However, an analysis via a passive lexical rule is impossible when the predicational heads are realised as two or more separate tokens. This is because the passive morpheme is morphologically attached to only one of the items in the verbal sequence and would naturally apply to only the subcategorisation frame of that item (Çetinoğlu, 2009). However, passivisation needs to be applied to the composed subcategorisation frame of the complex predicate (which is distributed across two separate tokens — *vangu* and *koṭu* — in this case).

Therefore, the existing lexical rule treatment of passivisation has been modified and instead, an analysis in terms of the Restriction Operator was developed (cf. also Wedekind and Orsnes (2003)). The advantage of this analysis is that all operations or subcategorisation frames are now treated via the same mechanism, and predicate composition can be treated in the same way irrespective of how the parts of a complex predication are realised: as two separate tokens or as a single complex verb, expressing the intuition that morphological and syntactic complex prediction involves the same mechanism (Alsina and Joshi, 1991). In the modified passivisation rule, we first restrict everything and then define the passive transformations.

The passivisation template below in (32) shows how the Restriction Operator is integrated into the PASSIVE frame. The template (32) ensures that passivisation is applied to the whole complex predicate and not only to the light verb. Templates in XLE are used to define common functional patterns or generalisations that can be reused whenever necessary (Crouch et al., 2017). The PASS template takes one argument: an instantiated variable labeled "_SCHEMATA". This represents the subcategorisation frame of whatever verb is being dealt with (e.g. *vangu* 'buy'). The first disjunct of the PASS template just gives back the active version, and nothing happens except that the structure is marked as [PASSIVE −]. In the second disjunct, the structure is constrained to be passive, the SUBJ is suppressed (NULL) or is realised as an instrumental OBL (OBL-INST) and the OBJ is realised as the SUBJ.

(32)

PASS (_SCHEMATA) = {
_SCHEMATA "active version"
(↑ PASSIVE)= -
| _SCHEMATA "passive version"
↓ \PRED\SUBJ\CHECK\OBJ-TO\OBJ\PASSIVE =
↑ \PRED\SUBJ\CHECK\OBJ-TO\OBJ\PASSIVE
(↑ PASSIVE)=c +
{(↑ SUBJ) –> NULL
| (↑ SUBJ) –> (↑ OBL) }
(↑ OBJ) –> (↑ SUBJ) }

5.3.12 Causatives

A causative in Tamil can be realised either morphologically or syntactically (Nuhman, 1999; Paramasivam, 2011), as discussed in Chapter 4. However, in either case, the causative is realised as a monoclausal complex predication (Annamalai, 2013).

5.3.12.1 Morphological Realisation of Causatives

The example in (33) shows how the morpheme வி (vi) is used in causatives. As shown in (34), the f-structure for (33) analyses the morphological causative as a complex predicate (Butt, 2010; Butt et al., 2003); the argument roles in this causative are causer.NOM and causee.INST. The case marking of the patient is NOM. The causative morpheme co-predicates together with the main verb.

(33)

வாங்குவித்தான் (vāṅkuvittāṉ)				
வாங்கு	-வி	-த்	-த்	-ஆன்
vangu	-vi	-t	-t	-aan
buy	-CAUS	-SAN	-PAST	-3SMR

'he made somebody buy (something)'

(34)
$$\left[\text{PRED} \quad \text{`caus}\left\langle(\uparrow \text{SUBJ}), \text{`vangu}\left\langle(\uparrow \text{OBL-INST}),\ (\uparrow \text{OBJ})\right\rangle\text{'}\right\rangle\text{'}\right]$$

The example in (35) illustrates how a light verb and a causative can be stacked in the sense that the causative applies to the entire complex predicate construction, irrespective of whether the two predicational units are written together or separately. The corresponding f-structural analysis is shown in (36).[11]

(35)

நான்	அவளைக்கொண்டு	அவனுக்கு	
naan	avaḷaikkoṇṭu	avaṉukku	
I.NOM	she.ACC.INST	he.DAT	
ஒரு	கார்	வாங்கிக்	கொடுப்பித்தேன்
oru	car	vangi-k	koduppitthen
a	car.NOM	buy.VP-SAN	give.CAUS.PAST.1SG

'I got her to buy a car for him.'

[11] Causatives in Tamil require in-depth linguistic exploration. For instance, it is not clear why OBL-INST takes an accusative case marking before the instrumental marking. This accusative marking is done only when *koṇṭu*, a derived form of *koṭu*, is used as the instrumental marker.

(36)

$$\begin{bmatrix}
\text{PRED} & \text{`caus}\left\langle (\uparrow \text{SUBJ}), \text{`kodu}\left\langle (\uparrow \text{OBJ-TH}), \text{`vangu}\left\langle (\uparrow \text{OBL-INST}), (\uparrow \text{OBJ})\right\rangle\text{'}\right\rangle\text{'}\right\rangle\text{'} \\
\text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{`PRO'} \\ \text{PRON-FORM} & \text{naan} \\ \text{CASE} & \text{NOM} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 1 \end{bmatrix} \\
\text{OBJ-TH} & \begin{bmatrix} \text{PRED} & \text{`PRO'} \\ \text{PRON-FORM} & \text{avan} \\ \text{CASE} & \text{DAT} \end{bmatrix} \\
\text{OBL-INST} & \begin{bmatrix} \text{PRED} & \text{`PRO'} \\ \text{PRON-FORM} & \text{aval} \\ \text{CASE} & \text{INST} \end{bmatrix} \\
\text{OBJ} & \begin{bmatrix} \text{PRED} & \text{`car'} \\ \text{CASE} & \text{NOM} \\ \text{DEF} & - \end{bmatrix} \\
\text{TNS-ASP} & \begin{bmatrix} \text{TENSE} & \text{PAST} \\ \text{MOOD} & \text{INDICATIVE} \end{bmatrix} \\
\text{VTYPE} & \left[\text{COMPLEX-PRED} \quad \text{VV}\right] \\
\text{PASSIVE} & - \\
\text{STMT-TYPE} & \text{decl}
\end{bmatrix}$$

5.3.12.2 Syntactic Realisation of Causatives

As discussed in Chapter 4, causatives in Tamil can also be realised syntactically by adding one of the following verbs after an infinitive form of the main verb: செய் (sei) 'do', வை (vai) 'put', பண்ணு *pannu* 'do'. These verbs have the character of light verbs, expressing the meaning 'make'. When one of these three verbs is used as the main verb of predication, the other two verbs function as causativising light verbs in the combination. As shown in (37), we again face a by now well-known implementational challenge: the main verb and the causative light verb can be written together as one token or separately as two. Further, ப் (p) or ச் (c) will be added as a *Sandhi* to the main verb as part of the causativisation, for *pannu* 'do' and *sei* 'do', respectively. There is no *Sandhi* for *vai* 'put' as it begins with a consonant வ்(v).

(37)

அவனை	ஒரு	கார்	வாங்கச்	செய்தேன்
avan-ai	oru	car	vanga-c	seithen
he-ACC	a	car.NOM	buy.INF-SAN	make.PAST.3SM

'(I) made him buy a car.'

5.3.13 Coordination

Conjunction and disjunction in Tamil can be realised morphologically or syntactically, as discussed in Chapter 4, in addition to the use of punctuation. The *-um* suffix has several functions, including conjunction and focus-marking, both of which are available in my corpora. I used the METARULEMACRO macro to handle coordination. The METARULEMACRO is a special macro defined in ParGram,[12] and if this macro is present in grammar, it is applied to all of the rules in the grammar. The reason to use this is that when we add in new rules, we do not have to remember to add in calls to other rules or macros that should apply to them, such as coordination. Instead, XLE will do this automatically for us. This particular macro is used in most ParGram grammars to handle coordination.[13] Therefore, I have also used the METARULEMACRO to handle NP coordination and Adjectival phrase coordination without much hassle.

(38) is one such coordination example, and its analysis is shown in Figures 5.19 and 5.20. Here comma is used to conjoin NPs. However, the very last NP — cassava — takes on the *-um* suffix. In this analysis, I consider *-um* as a focus marker. That *-um* takes a focus function can be observed from certain constructions available in my corpora which only involve one NP that includes a *-um* suffix; in this case, *-um* is a focus marker. In (38), I maintain a similar analysis, where I consider *-um* as a focus marker since I take the coordination to be achieved through commas.[14]

[12] `https://ling.sprachwiss.uni-konstanz.de/pages/xle/doc/PargramStarterGrammar/eng-pargram.lfg`

[13] `https://ling.sprachwiss.uni-konstanz.de/pages/xle/doc/PargramStarterGrammar/starternotes.html`

[14] One can still argue that *-um* functions as a concatenation operator that applies to all preceding entities to conjoin them. In general, the *-um* suffix and the coordination require further exploration.

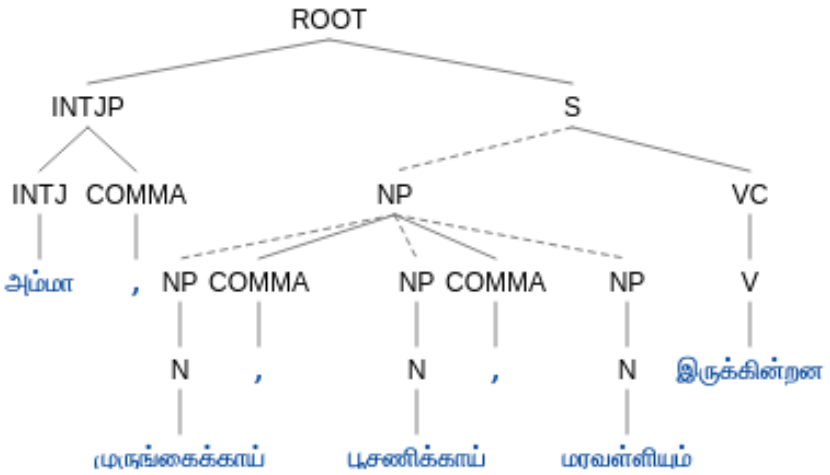

Figure 5.19: C-structure analysis of Example (38)- A Coordination example

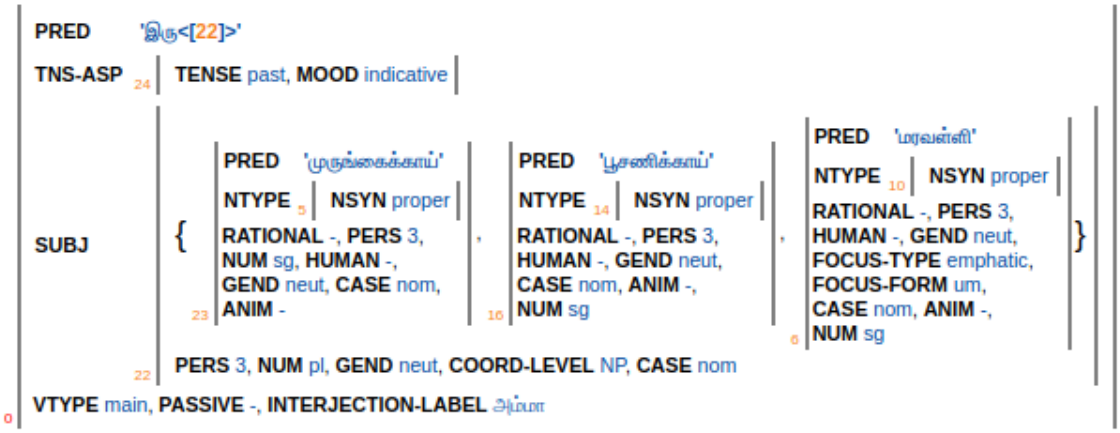

Figure 5.20: F-structure analysis of (38)

(38)

அம்மா ,	முருங்கைக்காய் ,	பூசணிக்காய் ,	மரவள்ளியும்	இருக்கின்றன
am'mā,	muruṅkaikkāy,	pūcaṇikkāy,	maravaḷḷiy-um	irukkinrana
mother,	drum-stick,	pumpkin,	cassava-and	exist.PRES.3PLN

‘‘“Mother, drumstick, pumpkin, cassava-and exist (i.e are available)”’

5.4 LFG Treebank

In addition to the grammar, I have also created a treebank with LFG annotations that is now available on the INESS treebank repository. Currently, it has 100 sentences taken from the ParGram project and Grade-1 textbook with LFG annotations. In order to

generate this treebank, I first parsed a sentence, and then I manually disambiguated generated parses to get the correct one/s. I then exported the parsed output as a Prolog file that was then uploaded to the INESS treebank repository.

5.5 Evaluation and Discussion

My corpora have 100 sentences from the ParGram project corpus and 100 sentences from a Grade-1 educational Tamil textbook, which is meant for native Tamil students aged five. Section 5.3 provided an account of the different types of constructions that the grammar I have developed can handle. This analysis still does not cover all of the constructions in my corpora, especially due to a lack of linguistic clarity regarding certain constructions. Verbal or gerundial nouns, serial verbs, and suffixes like *-um* are some areas or concepts that require deep linguistic exploration. For instance, in (39), *iruppataik* and *pe̲r̲ravar* are gerundial nouns that can function as nominalised verbs. These show mixed properties (Butt et al., 2020), specifically, both nominal and verbal properties. For instance, *iruppataik* shows nominal properties by taking an accusative case marking and verbal properties by taking an OBL argument. Similarly, *pe̲r̲ravar* takes nominative case marking, and licenses for the OBJ — *paṭam*. These Tamil structures require more linguistic exploration and studies on how they can be analysed within the LFG formalism.

(39)

படம்	பெற்றவர்	அதில்	இருப்பதைக்	கூறுங்கள்
paṭam	pe̲r̲ravar	atil	iruppataik	kūr̲uṅkaḷ
picture	receiver.NOM	it.LOC	do-exist.ACC	say.2PL

'Persons who received the picture say what is in it'

In general, I found that the grammar handles free-word order well and parses all valid constructions formed using the lexical items found in the lexicon. I have not tried sentence generation yet as I have been using INESS interface for grammar writing that does not support generation. However, I will try with XLE toolkit to see whether I can do generation in the future. If that works, it will be a good asset for generating artificial data for machine learning purposes.

The grammar rules will need to be optimised, and more constraints need to be added to reduce the number of possible analyses. For instance, certain complex constructions generated 20 solutions that needed to be manually disambiguated. One main reason for these generations of all solutions is our flexibility in CASE marking and word order. As I discussed in Chapter 4 and Section 5.3, nouns are not always overtly marked for case; still, they can function differently. For instance, irrational nominative entities can always be confused when they do not take object case marking, which marking is in turn not required for indefinite irrational nouns. The case syncretism in Tamil adds

another level of ambiguity; dative subjects, indirect objects and benefactive objects are always confusing. Tamil also has nominal modifiers, in which two nouns marked for the nominative case can modify one another. At times, there are no syntactic clues with which to decide whether nouns are modifying one another.

Pro-drop and elision, which are very common in Tamil, add another level of complexity in defining constraints. For instance, consider (40). One of the available readings is that the tractor has been bought for the farmer's son (where SUBJ is dropped, and genitive marking on the farmer is elided) or that the farmer is the SUBJ of the buy-give complex predicate. Nevertheless, this can also be analysed where 'farmer' is the subject of the construction. Writing rules to solve this ambiguity requires a finer exploration of how modification works and how different associated interactions can be captured with LFG.

(40)

விவசாயி	மகனுக்கு	ஓர்	உழவு	இயந்திரத்தை	வாங்கிக்கொடுத்தார்
vivacāyi	makaṉukku	ōr	uḻavu	iyantirattai	vāṅkikkoṭuttār
farmer	son.DAT	a	tillage	machine.ACC	buy-give.PAST.3SE

'The farmer bought his son a tractor / (someone) bought a tractor for farmer's son'

5.6 Conclusion

This chapter outlines how I developed the first rule-based grammar for Tamil using Lexical-Function Grammar. This is the only rule-based Tamil grammar publicly available for others to use and build upon. The current grammar version can handle common syntactic constructions found in Tamil using the given vocabulary, including the different word orders of these constructions. The grammar is helpful for deep syntactic analyses of the Tamil language. As I have set out to show in Section 5.3, very detailed syntactic information can be obtained from the grammar, even if other interesting constructions can be handled with the current grammar, as it is much wider and can parse more constructions. In order to provide more precise syntactic information, I have added features to the ones proposed in the ParGram project specification.

In order to increase the usage of this grammar, I have written the grammar using the sentences from the ParGram project using the ParGram specification. With this, one can now do a cross-lingual study by comparing the parses of Tamil with other languages that are documented as part of the ParGram initiative. In addition to the ParGram sentences, I have also used sentences from a native Tamil textbook so that I can capture the native structures in Tamil.

The current version of the LFG parser can be accessed via the INESS platform.[15] However, to use the parser, one needs to know the vocabulary list. Therefore, I have also developed a web interface to form sentences based on the available vocabulary and then parse them using the parser. I am confident that this grammar would be a useful resource for people who would like to do deep syntactic analyses to understand different Tamil constructions or carry out cross-linguistic studies.

I have made the following contributions with respect to rule-based grammar development:

1. created the first rule-based Tamil grammar using Lexical-Functional grammar formalism. This grammar is the only publicly available rule-based grammar with significant coverage.

2. published a treebank annotated according to the ParGram specification

3. shed light on complex predicates in Tamil based upon the grammar that has been developed, with the output of that having also been published in Sarveswaran and Butt (2019).

5.7 Future work

This work can be further extended computationally and linguistically. From the computational point of view, this grammar can be further optimised by adding further features. Although I have made an initial attempt to integrate the morphological analyser into the grammar, more work still needs to be done. Integrating morphological guessers may improve the robustness of the grammar. While I have been using the default tokeniser for pre-processing, one can invest further time in this in order to clean and prepare inputs. Moreover, one can look into the idea of multi-word tokenisation proposed in the Universal Dependencies and see whether adapting such multi-word tokenisation would make grammar writing easy and help to capture more syntactic information. I have also tried to develop a bridge between LFG and the Universal Dependencies so that I can migrate the treebank I generate in LFG to the Universal Dependencies to make more use of LFG grammar. This is, however, yet to be completed.

More broadly, much more work needs to be done from a pure linguistic angle, especially given the lack of deeper and more analytic yet less confusing studies on certain aspects of Tamil grammar. For instance, complementation, nominalisation, complex predication, syntactic properties of clitic like *-um*, postposition, *etc.* all require further in-depth research. A study on Tamil language change and the grammaticalisation of different words is also necessary in order to better understand certain constructions.

[15] `https://clarino.uib.no/iness/xle-web`

6 DATA-DRIVEN PARSING

This chapter describes how I have developed a data-driven parser using the Universal Dependencies (UD) framework. The chapter includes a background to the UD, the data-preparation approach, the training and evaluation of the parser, and a conclusion. In addition, the chapter also includes a detailed discussion on the analysis of different constructions using UD.

6.1 Universal Dependencies (UD)

The Universal Dependencies (UD) is a framework used for consistently annotating Parts of Speech, morphological features, and syntactic dependencies across human languages. UD has been evolving as an open community effort producing nearly 202 treebanks in over 114 languages as of May 2021. The main goal of this project is to facilitate multi-lingual parser development, cross-lingual learning, and parsing research from a typological perspective.[1] The UD annotation scheme has been evolving since 2014 on the basis of the following key principles (de Marneffe et al., 2021):

- it has to be cross-linguistically consistent;
- it scheme should facilitate multi-lingual research studies;
- it should as much as possible be derived from existing standards.

Since its initial release in 2014, the specification has evolved and been extended to accommodate new languages and new features within the existing languages.

The UD annotation follows a lexicalist view of syntax, where dependency relations hold between syntactic words. Other information such as morphology, POS and lemma information are given as properties for each word. Since dependency relations are established between syntactic words, words with more than one set of associated syntactic information or words that a single POS cannot adequately represent are broken into separate words or individual syntactic units before the dependency relations are established. Tokens with more than one set of syntactic information are called multi-word tokens, and the process which uses these multi-words and then separates

[1] https://universaldependencies.org/introduction.html

them in accordance with their syntactic units is called multi-word expansion (de Marneffe et al., 2021). For instance, clitics are considered multi-word tokens and are, in most cases, broken up, i.e. segmented as shown in (1), so that syntactic words and their dependency information can be marked clearly. For instance, in this example, since the multi-word token is segmented into syntactic words, we can identify different syntactic elements such as predicate, object, and beneficiary and can be able to mark the dependency among them.

However, UD does not lose the orthographic structure of the multi-word in the original text. In addition to the expanded syntactic tokens, the original orthographic form is also included in annotations; see (2), for example. This information is all captured in the format presented in (2) (see Table 6.1 for the information about each column, which will also be discussed in later sections). In (2), *it's* in one line is expanded to multiple lines, as *it* and *s*. This multi-word expansion constitutes an important phase for Tamil since words in the language are frequently concatenated in Tamil. In older times, Tamil was written without any spaces, and it was up to readers to decide where to pause or break.

(1) *dámelo*[2] 'give it to me' ==> *da me lo* (Spanish)

(2)

#Text = 'It's just fun'

1-2	It's	_	_	_	_	_	_	_	_
1	It	it	PRON	PRP	Case=Nom| Gender=Neut| Number=Sing| Person=3| PronType=Prs	4	nsubj	4:nsubj	_
2	's	be	AUX	VBZ	Mood=Ind| Number=Sing| Person=3| Tense=Pres| VerbForm=Fin	4	cop	4:cop	_
3	just	just	ADV	RB	_	4	advmod	4:advmod	_
4	fun	fun	ADJ	JJ	_	0	root	0:root	_

All these annotations are documented in a format called CoNLL-U. CoNLL-U is a derived version of an annotation format proposed in the Conference on Natural Language Learning - 2010 called CoNLL-X. The annotation scheme has ten fields, as shown in Table 6.1. If a field value is null, this is filled with an underscore ('_').

[2] `https://universaldependencies.org/u/overview/tokenization.html`

Table 6.1: CoNLL-U annotation style - used in UD treebanks

Field/Column	Description
ID	Word index - an integer reset to 1 for each new sentence
FORM	Word form / punctuation symbol
LEMMA	Lemma / stem of word form
UPOS	Universal POS tag
XPOS	Language-specific POS tag
FEATS	List of morphological features
HEAD	Head of the current word - either a value of ID or zero (0).
DEPREL	Universal dependencies relation to the HEAD - 'root' iff HEAD=0
DEPS	Enhanced dependency graph - a list of head-deprel pairs.
MISC	Any other annotation - transliteration etc

UD treebanks represent different genres of text, including spoken corpora, transliterated corpora, code-mixed corpora, and others. The largest UD treebank in the current version is German HDT, which has around 3.5 million tokens annotated with POS, morphology and syntactic information. However, most other treebanks are small; for instance, Tamil has around 12K tokens. There are only three Indic languages: Hindi, Urdu, and Sanskrit, each with a relatively large dataset: 375K, 138K, and 28K tokens, respectively, in UDv2.8. All the other six Indic languages, including Tamil and Telugu, have less than 12K tokens in UDv2.8.

6.2 Levels of annotation in UD

6.2.1 POS annotation

UD uses a derived POS tagset from Google's Universal POS (UPOS) tagset (Petrov et al., 2012). This tagset consists of 17 tags which have been elaborated upon in Chapter 2. The challenge in POS annotation is capturing the context of words. As in other languages, the context change in Tamil can change the grammatical category of a word. For instance, words such as ஆண்டாள் (āṇṭāḷ) 'a proper name' or '(she) ruled' and படி (paṭi) 'step' or 'study.imperative' can be nouns or verbs depending on the context. Therefore, POS taggers should capture as enough information about different usage of these kinds of words as possible. This UPOS is used to annotate only the high-level grammatical categories. The finer levels of information is captured using morphological features.

6.2.2 Morphological annotation

Morphological annotations in the UD represent the contextual lexical and grammatical properties of words. Morphological annotations are stored as key-value pairs, as inspired by the format in Prague Dependency treebank (Böhmová et al., 2003; Zeman, 2008). The morphology labels are derived from the UniMorph project (McCarthy et al., 2018). UniMorph (Kirov et al., 2016) or Universal Morphology is a Universal annotation scheme for morphology that renders morphological feature labels for inflectional morphology. It is claimed that the inflectional morphology of any language can be defined using this scheme (Sylak-Glassman, 2016). However, I have shown in Chapter 3 that this is not true when it comes to Tamil. It does not, for instance, capture the key feature called *Sandhi*. The morphological scheme in UD has been revised regularly to accommodate new morphosyntactic features. In addition to these documented features, one can introduce any of the language-specific features to capture new language-specific morphosyntactic features if required.

6.2.3 Syntactic annotation

Syntactic information is the crucial annotation scheme in UD and is what is required for syntactic parsing. According to the annotation scheme, content words such as nouns, verbs, adjectives, and adverbs are associated with syntactic relations. The rest are modifiers connected with respect to content words via functional relations. Relations between words always start from a predicate, the root of the clause or sentence. Other content words and function words are connected with the use of a directed arrow starting from the root (de Marneffe et al., 2021). In UD, propositions are also considered modifiers or function words and are linked to respective content words using the label called 'case'.

Annotation in UD always starts with a predicate, and it is marked as the Head of the construction. All other constituents are marked as directly or indirectly dependent on the Head. In UD, the predicate can be either nominal or verbal, unlike formalisms such as AnnCorra where the predicate has to be a verb. Punctuations are also attached to the Head as a dependent. The UD borrows the annotation scheme from Stanford's syntactic relations (de Marneffe et al., 2006) to mark syntactic relations. There are 37 universal syntactic relations specified in UD version 2.8. These are listed in Figure 6.2. The relations are a closed set, yet they can be extended in different ways. For instance, relative clauses can be marked as *acl:relcl* and pre-determiners are marked as *det:predet*. However, in my research, I have not made use of EUD as this scheme came recently, and the Tamil language still requires further research to be pursued before exploring these extended features.

Table 6.2: The list of dependency relations in Universal Dependencies (de Marneffe et al., 2021)

Dependency relation	**Description**
acl	adnominal clause; finite or non-finite clause modifying a nominal.
advcl	adverbial clause modifying a predicate or modifier word.
advmod	adverb or adverbial phrase was modifying a predicate or modifier word.
amod	adjectival modifier of a nominal.
appos	appositional modifier; a nominal used to define, name, or describe the referent of a preceding nominal.
aux	auxiliary; links a function word expressing tense, mood, aspect, voice, or evidentiality to a predicate.
case	links a case-marking element (preposition, postposition, or clitic) to a nominal.
cc	links coordinating conjunction to the following conjunct.
ccomp	clausal complement of a verb or adjective without an obligatorily controlled subject.
clf	(numeral) classifier; a word reflecting a conceptual classification of nouns linked to a numeric modifier or determiner.
compound	any kind of word-level compounding (noun compound, serial verb, phrasal verb).
conj	conjunct; links two elements that are conjoined.
cop	copula; links a function word used to connect a subject and a nonverbal predicate to the nonverbal predicate.
csubj	clausal syntactic subject of a predicate
dep	unspecified dependency, used when a more precise relation cannot be determined.
det	determiner (article, demonstrative, etc.) in a nominal.
discourse	discourse element (interjection, filler, or non-adverbial discourse marker).
dislocated	a peripheral (initial or final) nominal in a clause that does not fill a regular role of the predicate but has roles such as topic or afterthought.
expl	expletive; links a pronominal form in a core argument position but which is not assigned any semantic role by the predicate.
fixed	fixed multi-word expression; links elements of grammaticalised expressions that behave as function words or short adverbials.

flat	flat multi-word expression; links elements of headless semi-fixed multi-word expressions such as names.
goes with	links parts of a word that are separated but should go together according to standard orthography or linguistic wordhood.
iobj	indirect object; nominal core argument of a verb that is not its subject or (direct) object.
list	links elements of comparable items interpreted as a list.
mark	marker; links a function word marking a clause as subordinate to the predicate of the clause.
nmod	nominal modifier; a nominal modifying another nominal.
nummod	numeric modifier; numeral in a nominal.
nsubj	nominal subject; a nominal core argument which is the syntactic subject (or pivot) of a predicate.
obj	object; the core argument nominal, which is the most basic core argument that is not the subject, is typically the most directly affected participant.
obl	oblique; a nominal functioning as a non-core (oblique) modifier of a predicate.
orphan	links orphaned dependents of an elided predicate.
parataxis	links constituents placed side by side with no explicit coordination or subordination.
punct	punctuation attached to the Head of its clause or phrase.
reparandum	repair of a (normally spoken language) disfluency.
root	root of the sentence
vocative	nominal directed to an addressee.
xcomp	clausal complement of a verb or adjective with an obligatorily controlled subject.

6.3 Tamil Universal Dependencies Treebanks

Tamil has been included in UD treebank releases since 2015. The initial version was populated from the Prague Style Tamil (PDT) treebank (Ramasamy and Žabokrtský, 2012), and since then, the dataset has been part of the UD. Tamil-TTB[3] in UDv2.6 has tokenisation and tagging inaccuracies and inconsistencies. For instance:

[3]`https://github.com/UniversalDependencies/UD_Tamil-TTB/tree/master`

- certain words are segmented incorrectly:
  - என்னிடம் *eṉṉiṭam* 'I.LOC' is tokenised to என்ன் + இடம். Segmenting case markers is unnecessary as case information is captured under morphological features. Further, in this case, the segmentation is also done in the wrong way. Similarly, several other case markers are also segmented and segmented in the wrong way.
  - செல்லலாம் *cellalām* 'can-go' is segmented as செல்லல் *cellal* 'act of going' + ஆம் *ām* 'yes'. This is also incorrect. There are similar segmentations with verbs that can be found in Tamil-TTB.
  - tokens are not segmented in the right place in some cases. For instance, வயதான *vayatāṉa* 'age-adjectivisor' has been broken up as வயத் **vayat* + ஆன *āṉa* 'become' . Because, வயத் **vayat* does not mean anything in Tamil, it has to be வயது *vayatu* 'age'.
- nouns in Tamil with dative case marking can be used to mark a subject, oblique and indirect object; this case marking is misused to mark an object in the treebank in most cases

I corrected some of these issues and made them available in UDv2.7. However, there are still more issues that need to be solved. Tamil-TTB in UDv2.6 has sentences for training, development and testing. Further, Tamil-TTB has very long sentences; some of them are even 40 words long sentences.

In addition to Tamil-TTB, together with scholars at the University of Hyderabad, India, I created a new treebank in 2020 called Modern Written Tamil Treebank (MWTT). Tamil-MWTT is now part of the UDv2.7 (Zeman et al., 2020) release. Tamil-MWTT has 534 well-structured sentences taken from a book called Modern Tamil Grammar (Lehmann, 1993), covering most of the single clause sentence constructions. We took sentences from the textbook so that there would be less ambiguity regarding the structure, which is good for us to kick start the annotation.

6.4 Tools for syntactic parsing

This section gives an account of existing parsing tools that are widely used when developing dependency parsers. I will specifically discuss three such tools, namely Stanza (Qi et al., 2020), TranKit (Nguyen et al., 2021), and uuparser (de Lhoneux et al., 2017a) as these are the ones I have used specifically for my experiments.

Zeman et al. (2017) and Zeman et al. (2018) have shown the evaluation of 33 and 17 tools, respectively, which were submitted to shared tasks. From the results, it is clear that most of these tools are built on top of a tool called UDPipe, and do not

consist of components with which to do all the annotations as required for the Universal Dependencies treebank. In the 2017 shared task, Stanza (Qi et al., 2020) showed the top performance for parsing, and in the 2018 edition, it performed better when there was a significant amount of data. It does not support multi-lingual learning. On the other hand, uuparser (de Lhoneux et al., 2017a) showed comparable performance to Stanza in 2018 shared task (Zeman et al., 2018) supports for multi-lingual learning. However, the uuparser does not do POS, Morphology and Lemma annotations. A recent arrival in this respect is TranKit, a transformer-based natural language toolkit.

6.4.1 Multi-lingual learning

Multi-lingual learning is an approach primarily used in the context of low-resource language development. The major obstacle in the development of tools and systems for a low-resource language is the lack of annotated resources. Therefore, multi-lingual learning is used to transfer linguistic knowledge between high-resource language and low-resource language. The key assumption in this approach is that specific distributional linguistic properties, language relatedness, script closeness, and geographical closeness (de Lhoneux et al., 2017a) are invariant across languages. This approach has been used in syntactic parsing as well. However, this has not yet been used for syntactic parsing of the Tamil language. In this research, I have also tried to make use of multi-lingual training in order to see whether this improves the parsing score in Tamil.

6.4.2 uuparser

uuparser is a syntactic parsing tool (Kiperwasser and Goldberg, 2016; de Lhoneux et al., 2017a,b) that does not use POS, morphological features, or lemmas to parse text, although these sets of information are favoured in pipeline-based parsing development approaches, which are followed in most of the other parsing tools. uuparser is a deep neural network-based tool that does parsing in two steps: 1. sentence and word segmentation, and 2. transition-based approaches to predict the dependency tree. Sentence/word segmentation has been modelled as a character-level sequence labelling problem in a Bi-RNN-CRF model. This model simultaneously predicts sentence and word boundaries and identifies multi-word tokens. The transition-based approach relies on a BiLSTM to learn information features of words in context. A feed-forward network is then used to predict the next parsing transition. These two approaches have been discussed in detail in de Lhoneux et al. (2017a). The latest version of uuparser also supports graph-based parsing.

6.4.3 Stanza

Stanza is a natural language toolkit consisting of tokenisation tools, POS cum morphological parsing, lemmatisation, dependency parsing and a Named Entity Recognition (NER). Among the NLP toolkits available until April 2021, Stanza was the only fully neural-based toolkit that provided all the features required to process raw text. It uses a Bi-LSTM-based deep biaffine neural network to implement dependency parsing using a transition-based approach (Qi et al., 2020). I have used Stanza to do everything from tokenisation to parsing. Chapter 2 outlines how Stanza has been used to implement a POS tagger. Stanza does both POS tagging and morphological analysis together. I have, however, handled these separately and discussed these two processes and developments in Chapter 2 and Chapter 3, respectively.

6.4.4 TranKit

TranKit (Nguyen et al., 2021) is a natural language toolkit introduced in April 2021 that uses multi-lingual transformer language models to implement parsing. This is a development based on the concepts and algorithms proposed in Stanza, and it even borrows Stanza's lemmatiser to do the lemmatisation. However, unlike Stanza, TranKit performs POS, Morphology and Dependency parsing together as one task. Since this toolkit was published very recently, I could not do any detailed evaluation of it.

6.5 Evaluation metrics for data-driven parsers

Several evaluation metrics evaluating different aspects of tagging or annotations are being used to evaluate parsers. Among these, there are four approaches that are being used widely in papers and shared tasks, namely: UAS (Unlabelled Attachment Score), LAS (Labelled Attachment Score), MLAS (Morphology-aware Labelled Attachment Score), and BLEX (Bi-Lexical dependency score). All of these metrics reflect word segmentation and relations between content words. UAS and LAS also include relations between other words but do not consider morphology and lemmas when calculating the final score. Since my focus is on evaluating dependency relations, UAS and LAS have primarily been used to evaluate proposed parsers.

Labelled Attachment Score (LAS) is a widely used evaluation metric in dependency parsing. This metric evaluates the percentage of words assigned both the correct syntactic Head and the correct dependency label. For scoring purposes, only universal dependencies labels will be taken into account, which means that subtypes such as acl:relcl (relative clause) will be truncated to acl both in the gold standard and in the parser output during the evaluation. All nodes, including punctuation, will be reflected

in LAS. This metric also takes word segmentation mismatches into account. Therefore, a dependency is scored as correct only if both nodes and relationships match the existing test set or gold-standard nodes. Precision P is the number of correct relations divided by the number of system-produced nodes; recall R is the number of correct relations divided by the number of test-set or gold-standard nodes. LAS is defined as the F1 score = 2PR / (P+R).

Unlabelled Attachment Score (UAS) is also a widely used evaluation metric in dependency parsing. Unlike LAS, in UAS, the percentage of words assigned with the correct syntactic Head is considered for evaluation. All nodes, including punctuation, will be counted for UAS. This metric also takes word segmentation mismatches into account. A dependency is scored as correct only if both nodes of the relationship match existing gold-standard nodes. Precision P is, therefore, the number of correct relations divided by the number of system-produced nodes; recall R is the number of correct relations divided by the number of test-set or gold-standard nodes. UAS is defined as the F1 score = 2PR / (P+R).

6.6 Design choices

6.6.1 Approach

The deep learning approach is state-of-the-art for natural language application development, including the development of syntactic parsers. This approach has also been used in shared tasks and competitions. Language models that are trained using millions of sentences give very accurate contextual information of the language data. Some of these multi-lingual models are trained with hundreds of languages together, and these models have proven to have better scores. All these are required for parsing, especially for the parsing of morphologically rich and relatively high free-word order language (Futrell et al., 2015) such as Tamil. Therefore, I have made use of a deep learning approach to develop my data-driven parser.

We do not need to develop these deep learning-based systems from scratch. Libraries such as PyTorch, TensorFlow, Keras and toolkits such as Stanza, NLTK, and TranKit make the implementation easy and fast. Among these, Stanza and TranKit have all the required tools, namely tokenisation, a POS tagger, a Morphological analyser, and a Named Entity Recogniser with which to develop a parser. I made use of the Stanza toolkit for the development of my Tamil dependency parser — *Thamizhi*UDparser. In addition, I also used uuparser to carry out multi-lingual training, given the insufficient Tamil data with which to train a parser. Section 6.7 outlines how all these toolkits are used to build a parser for the Tamil language.

6.6.2 Dataset

Training a deep learning-based system requires a significant amount of data. Transfer learning, data augmentation, and multi-lingual learning are some of the approaches used when there is insufficient training data. Since this research aims to build a deep syntactic parser, it is important to ensure that all required syntactic structures are captured and correctly annotated. This is not always possible with transfer learning and multi-lingual learning techniques, which will not capture language-specific constructions. Therefore, I decided to compile and annotate a dataset manually and then extend it with the use of multi-lingual learning along with Telugu, Hindi, and Turkish.

I first compiled a news corpus consisting of news text from different web pages belonging to 18 different genres as derived from the British National Corpus (Consortium, 2007) annotation scheme. I crawled only the text published after the year 2000 to ensure that the text contained only Modern Written Tamil text. Since Tamil sentences are usually long, and the handling of long sentences is challenging, the task becomes even more challenging if the language has free-word order. Further, more data is needed in order to train a parser to handle long-distance dependencies. Therefore, specifically for my research, I extracted and used sentences with a maximum of 15 tokens. The following issues have been identified during the tokenisation and cleaning phase of the process:

- while Tamil Unicode has become the de-facto standard for text encoding, some people, including publishing agencies, still use ASCII encoding. For this reason, all the text had to be checked for ASCII encoding.

- the shapes of some Tamil number symbols and the Tamil alphabet may look similar to those unaware of Tamil numbers. Therefore, there is sometimes a mix of both symbols and alphabet, and these have different code points in Unicode space and are considered as different characters by computers. All the text thus needed to be checked for this kind of glyph issue.

- due to the nature of the Tamil Unicode encoding and input methods, all the inputs are needed to be Unicode normalised before being fed to the web interface for analysis or generation. Multiple code sequences can form the same character in Tamil if the keyboard input driver is not controlled or handled. For instance, the letter கோ can be entered by users using the following sequences: க + ொ + ா or க + ொ + ◌ா . However, it is only the first sequence that is acceptable and logical, as in Tamil, a composite character such as கோ is formed by adding a vowel to a consonant. In Unicode, vowels are denoted by vowel modifiers. Therefore, a consonant cannot be followed by two vowel modifiers. However, in the case of க + ொ + ◌ா , the two vowel modifiers are followed by a consonant,

and this is thus impossible in Tamil. It is for this reason important to convert all the unacceptable formations to acceptable formations; the process of converting other forms to Unicode normalised form is called Unicode normalisation.

Machine learning systems require three datasets. These are usually non-overlapping sets, namely training, validation and testing. The data I crawled from the Web is primarily used for training and validation.

The new dataset collected was cleaned and annotated with POS, Morphological and Dependency information according to the Universal Dependencies version 2.0 scheme. As outlined in Section 6.7, I have used a development pipeline with which to do the annotation. Table 6.3 gives an outline of the size of data, source *etc*, and the purpose for which each dataset is used.

Table 6.3: Datasets used to implement the syntactic parser

Dataset	**Source**	**Task to which it is used**	**no. of sentences**	**Description**
*Thamizhi*TB	News text	Training and Validation	1,000	Taken text from newspapers published after the year 2000
*Thamizhi*TB +Tamil-MWTT	Book	Development or Validation	275 + 135	Text are from a grammar book by Nuhman (1999)
*Thamizhi*TB +Tamil-MWTT	Book	Testing	275 + 400	Text are from a grammar book by Lehmann (1993)

For testing, it is always good to use a widely accepted public dataset that is carefully prepared. Unfortunately, no such datasets are available for Tamil, except Tamil-MWTT, which has recently been published.[4] Tamil-TTB has a lot of inconsistencies and errors, as discussed Section 6.3. Apart from sentences from Tamil-MWTT, I created another test dataset, called *ThamizhiTB*-Testing, based on data from a Tamil book called Basic Tamil Grammar by Nuhman (1999) which consists of more complex constructions than what is covered in Tamil-MWTT, including coordination and embedded constructions. However, I needed to have these annotated with the help of linguists and language experts.

ThamizhiTB-Training and *ThamizhiTB*-Testing consists of several types of syntactic constructions. *ThamizhiTB*-Testing, in particular, has most of the different syntactic constructions available in Tamil since these were part of all the examples present

[4] `https://github.com/UniversalDependencies/UD_Tamil-MWTT/tree/master`

in the modern Tamil grammar (Nuhman, 1999), which consists of the most common occurring constructions in modern Tamil, and has been used as a textbook for people who specialise in Tamil in their advanced level of studies in Sri Lanka. I have also made use of Tamil-MWTT, which has only single clause constructions.

There are different classifications of syntactic categories as proposed in the literature. For instance, Jurafsky and Martin (2017) propose declarative, imperative, yes-no question and wh-questions as possible classifications. Butt et al. (1999) listed high-level categories such as declarative, interrogative, imperative, and embedded clauses. The latter is categorised further in Butt et al.'s (1999) classification. I have proposed a categorisation shown in Table 6.4, considering the nature of Tamil syntax. It is also important to note that the high-level categories proposed here can be further broken down. For instance, the nominal predicate category consists of the following non-canonical subjects: NP_{NOM} + NP_{NOM}, NP_{DAT} + NP_{NOM}, NP_{NOM} + NP_{NOM} + COP, and NP_{LOC} + NP_{NOM}. This categorisation was useful to ensure that the parser captures most of the constructions found in Tamil and checks which types of constructions the parser fails. A more granular level will be useful for detailed analysis. However, preparing data covering such a granular level of constructions will require good linguistic clarity and significant effort.

Table 6.4: Categorisation of syntactic constructions

Syntactic category	**Description**
Nominal predicate	linking verbs or copula are optional in most common cases.
Intransitive sentences	verbal predicates with only one core argument.
Transitive sentences	transitive and di-transitive sentences are part of this category.
Coordinated sentences	both conjunctions and disjunctions that are realised syntactically or morphologically are part of this category.
Complex sentences	sentences with embedded clause/s.

Tamil is a *wh*-in-situ language where interrogatives are marked morphologically or syntactically using *wh*-pronouns. Interrogatives are, therefore not categorised separately. Instead, they have been included in other given categories. Similarly, *pro*-drop is very common in Tamil; this has, however, not been categorised as a separate category. We need to use the Extended Universal Dependencies (EUD) if we need to capture *pro*-drop and elisions in detail. Therefore, I will refrain from engaging further on this here.

6.7 Methodology

Most of the dependency parsers use information such as POS, morphology and lemma to determine dependency labels (Qi et al., 2020; Nguyen et al., 2021). A recent survey on unsupervised dependency parsing (Han et al., 2020) compares the parsers developed from 2004 until 2019 for English and concludes that the parsing results are not comparable to supervised approaches. According to this paper, to generate a dependency parse tree, significant linguistic information is required to determine the functions of each word. This information has so far not been captured best by using unsupervised approaches. If an unsupervised approach fails for structured languages such as English, then it will perform even worse for morphosyntactically complex free-word order languages such as Tamil. It is for this reason that I opted to make use of a supervised approach.

As discussed in this Chapter and previous chapters, Tamil does not have sufficient annotated data to train a parser. Annotating data from scratch requires significant person-hours, and it's a tedious task. Therefore, I first decided to train a parser step by step, using existing tools and resources, and I adopted the development pipeline used by Stanza (Qi et al., 2020), which is shown in Figure 6.1. Instead of training a parser end to end, training it step by step through a pipeline of this type is more useful for languages such as Tamil because there is not much dependency annotated data available publicly.

On the other hand, essential tools such as a POS tagger and a Morphological analyser are already available (Sarveswaran et al., 2021; Sarveswaran and Dias, 2021). Therefore, these can be used to prepare data if a step-by-step process is followed. Table 6.5 shows how different tools and resources were used in order to prepare data. Since the current release of the Stanza, toolkit does not have support for multi-lingual training, I also used uuparser with the multi-lingual training for dependency parsing to develop a dependency parser for Tamil. Each of the phases within the pipeline is explained in the following sub-sections.

I developed — *Thamizhi*UDparser — the first version of the Universal Dependencies parser as the baseline system. Because annotating data from scratch consumes much time, this initial version was trained using the Tamil-TTB treebank, even though this is not perfect training data. The development of the baseline system has been reported in Sarveswaran and Dias (2020). I later used this first version of *Thamizhi*UDparser to prepare data for training and evaluation sets. After that, these two machine-annotated datasets were verified manually by linguists. Finally, the corrected training and development data were used to train several dependency parser models using two training approaches, such as monolingual training and multi-lingual training, and two parsing algorithms, such as graph-based algorithm and transition-based algorithm. Eventually,

Figure 6.1: Phases of the Parsing Pipeline
Image source: https://stanfordnlp.github.io/stanza

Table 6.5: *Thamizhi*UDparser process pipeline

Step or task	**Tool**	**Dataset**
Tokenisation	Own scripts and Stanza toolkit	Tamil-TTB
Multi-word tokeniser	None (did manually)	Tamil-TTB
Lemmatisation	Stanza toolkit	Tamil-TTB
POS tagging	*Thamizhi*POSt	Amrita Data
Morphological analysis	*Thamizhi*Morph	News text and Word lists
Dependency parsing	uuparser	UD treebank of various languages

I found that the parser implemented using Stanza toolkit using monolingual training using transition-based parsing gave better results — this I consider as the version two of *Thamizhi*UDparser. Finally, I also evaluated version two of *Thamizhi*UDparser using a categorical evaluation dataset I prepared. This step-by-step process has been elaborated on in the following sections.

6.7.1 Tokenisation

First, the training, validation and testing datasets (except Tamil-MWTT) were Unicode normalised, tokenised, and then broken up into sentences. I have developed a script[5] to do Unicode normalisation. During this phase, punctuations were also separated from words. Then texts were broken down into sentences using the model trained using the Stanza toolkit. Finally, all sentences were converted to the CoNLL-U format, which is used to mark annotations in the UD treebanks.

[5] `https://github.com/sarves/thamizhi-validator`

Table 6.6: Multi-word tokens handled by the parser

Multiword Token	Example
N+*um*	*avaṉ+um* | he+CONJ | 'Also he'
N+*tāṉ*	*avaṉ+tāṉ* | he+EMPH | 'Only he'
N+*tāṉ+ē*	*avaṉ+tāṉ+ē* | he+EMPH+WHQ | 'Only he, isn't he?'
N+PSP	*avaṉ-ai+paṟṟi* | he-ACC+about | 'About him'
V+*tāṉ*	*va-nt-āṉ+tāṉ* | come-PST-3.SG.M.+EMPH| 'He only came'
V+*ā*	*va-nt-āṉ+ā* | come-PST-3.SG.M.+QUES | 'Did he come?'

+ = multiword boundary; V = Verb; N = Noun; PSP = postposition

6.7.2 Multi-word tokenisation

After the initial tokenisation, syntactically compound words or multi-word tokens were broken into syntactic units as discussed so that syntactic dependencies could be marked precisely, an important task for further cross-lingual analyses. Syntactically compound constructions are common in Tamil. Yet, not all of such compounds need to be segmented or expanded. This is because certain constructions, specifically V+V_{AUX} constructions, can be morphologically marked. In the current scope, constructions shown in Table 6.6 are considered multi-word tokens, and the training data were preprepared accordingly. Multi-word expansion necessitates further research.

6.7.3 Lemmatisation

UD Treebanks also have lemmas marked in their CoNLL-U format. This is used to determine suitable dependency labels in Stanza. Therefore I made use of the Stanza toolkit to train a lemmatiser using the Tamil-TTB UDv2.5 dataset to do the initial annotation. Like in other stages, this dataset is also then verified manually.

6.7.4 POS tagging using *Thamizhi*POSt

In Chapter 2, I have discussed how the POS tagger *Thamizhi*POSt has been developed with the orchestration of various resources and tools. This has been used to conduct POS annotation in the CoNLL-U format, which already has lemma information. Even though *Thamizhi*POSt can be used to annotate data with the UPOS and Amrita tagset, the UPOS annotation has been used as it is the one used in Universal Dependencies treebanks.

6.7.5 Morphology annotation using *Thamizhi*Morph

I made use of a rule-based approach to do the morphological annotation, as outlined in Chapter 3. The tokens are tagged using the morphological features outlined in the

Universal Dependencies framework. This was later reviewed. As discussed in Chapter 3, since contextual tagging was not possible with rule-based tagging, all the annotations had to be manually reviewed once again to ensure that the morphological annotations were done correctly.

6.7.6 Dependency annotation using the version 1 of *Thamizhi*UDparser

The datasets, which is CoNLL-U formatted, now consist of POS, lemma, and morphological information. At this stage, dependency information can now be added to fulfil the UD annotation requirement. I made use of the UD 2.0 scheme for the annotation, which requires the following two pieces of information about dependency included in the CoNLL-U: 1. the token id of the Head of the given word and 2. the respective dependency label. Instead of doing this dependency annotation from scratch, I used version 1 of the *Thamizhi*UDparser to do the initial dependency annotations and then verified them manually. This section outlines this annotation process in detail and discusses how I annotated training, validation, and testing datasets.

There are existing trained models available for dependency parsing of Tamil. However, those models are trained on Tamil-TTB, which has issues as discussed. Therefore, I trained my own dependency parser on top of the POS and morphology annotations. Again to do the initial dependency annotations, I made use of the available Stanza pre-trained model[6] on the data I had annotated with POS and Morphology. Then I got the annotations fixed manually. With this annotated dataset, I made use of uuparser and multi-lingual training to develop a new dependency parser for Tamil — version 1 of *Thamizhi*UDparser— as the next step. This has been reported as a part of Sarveswaran and Dias (2020). I tried multi-lingual training along with Hindi-HDTB,[7] Turkish,[8] Arabic,[9] and Telugu,[10] which I found would be relevant, and which are available in UDv2.5.

Table 6.7 outlines the number of sentences used to do multi-lingual training and the results. As shown in the table, I made use of multi-lingual training with different combinations of languages and the number of sentences to develop different models. Then I tested all these models using the Tamil-TTB[11] test set, which is used by other parsers as well as for evaluation. This initial experiment shows better results when compared to all available parsers, which were built around UDv2.5. For instance, Stanza's pre-trained model gives a score of 57.64, and uuparser's pre-trained model shows a score of 55.76 for the Tamil-TTB test set.

[6] https://stanfordnlp.github.io/stanza/download_models.html
[7] https://github.com/UniversalDependencies/UD_Hindi-HDTB/tree/master
[8] https://github.com/UniversalDependencies/UD_Turkish-IMST/tree/master
[9] https://github.com/UniversalDependencies/UD_Arabic-PADT/tree/master
[10] https://github.com/UniversalDependencies/UD_Telugu-MTG/tree/master
[11] https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-3105

Table 6.7: LASs of Multi-lingual models

Languages	**(# of sent.)**	**(LAS)**
Telugu	100	58.91
Telugu	1050	59.22
Hindi	1600	**62.39**
Telugu Arabic	100 100	58.04
Telugu Turkish	100 100	58.43
Telugu Hindi	100) 100	59.07
Hindi Telugu	13,300 1050	60.65

I found that the parser performs better when trained with 1,600 sentences from Hindi-HDTB than when trained with Telugu, another Dravidian language. I have also trained the tagger with the whole Telugu and Hindi treebanks, along with Tamil. However, the score was less than what we got when we trained it with 1600 sentences from Hindi-HDTB. Telugu UDv2.5 is small in size and has around 1050 sentences when compared to Hindi-HDTB UDv2.5, which is 16,647 sentences. Moreover, the Telugu dataset had very short sentences without any morphological feature information in UDv2.5. On the other hand, some sentences in Tamil-TTB in UDv2.5 have more than 40 tokens. More importantly, Hindi-HDTB is a well-researched and quality dataset and includes of all the annotations.

Finally, I used version 1 of *Thamizhi*UDparser to add dependency annotations to training, validation, and testing datasets. I later had these annotated datasets reviewed by three people, two annotators and one trained linguist. Section 6.9 will discuss how I used these datasets to train a new set of dependency parsers, including version 2 of *Thamizhi*UDparser, which is implemented using the Stanza toolkit.

6.8 Analysis of datasets

This section describes the datasets used to train, evaluate, and test the data-driven parser. The datasets that are annotated using the CoNLL-U format have ten different data fields. Among these seven columns are the ones key for the UD annotation scheme 2.0, which I used to annotate data. These fields include token id, a surface form of the token, lemma, POS, morphology, dependency id, and dependency relation name, as shown in 6.8.

Table 6.8: Fields used in CoNLL-U data format

#Text = It's just fun

Token id	**Surface form**	**Lemma**	**POS**	**XPOS**	**Morph**	**Dep. id**	**Dep. name**		
1-2	It's	_	_	_	_	_	_	_	_
1	It	it	PRON	PRP	Case=Nom| Gender=Neut| Number=Sing| Person=3| PronType=Prs	4	nsubj	_	_
2	's	be	AUX	VBZ	Mood=Ind| Number=Sing| Person=3| Tense=Pres| VerbForm=Fin	4	cop	_	_
3	just	just	ADV	RB	_	4	advmod	_	_
4	fun	fun	ADJ	JJ	_	0	root	_	_

6.8.1 Part of Speech information

The Universal Dependencies uses the Universal POS tags, which include 17 tags. Table 6.9 shows that out of the 17 tags, 14 tags are used in my datasets. Interjection (INTJ), Symbols (SYM), and Unknown (X) POS are not present in the datasets. All other POS are otherwise found with good counts in the datasets, including subordinating conjunction. Only ten instances of particles (PART) are found; however, this is a closed category, and usually, not many particles are available in texts.

Table 6.9: Part of Speech tags used in the Tamil annotations

ADJ	257	NUM	364
ADP	179	PART	10
ADV	444	PRON	799
AUX	273	PROPN	576
CCONJ	151	PUNCT	1850
DET	169	SCONJ	77
NOUN	3443	VERB	1988

6.8.2 Morphological features

The datasets consist of the list of features shown in Table 6.10. The first column of the table shows the list of features, the second shows their associated values, and the final column shows the number of times the feature has been used in the datasets. In

addition to the features proposed by UD,[12] I have incorporated features such as *Sandhi* to capture Tamil morphology. UD uses the UniMorph (Kirov et al., 2016) to define morphological features. However, features such as *Sandhi* are missing in the UniMorph specification as well.

Table 6.10: Morphological features used in the datasets

Feature	**Counts**	**Feature value**	**Description**
AdpType	122	Post	adposition type postposition
Animacy	34	Anim	Animate
Aspect	143	Hab	Habitual
		Prog	Progressive
		Prosp	Prospective
Case	4606	Abl	Ablative
		Acc	Accusative
		Ben	Benefactive
		Com	Comitative
		Dat	Dative
		Gen	Genitive
		Ins	Instrumental
		Loc	Locative
		Nom	Nominative
Definite	21	Def	Definite
Deixis	225	Prox	Proximate
		Remt	Remote
Gender	5417	Com	Common gender
		Fem	Feminine
		Masc	Masculine
		Neut	Neuter
Mood	193	Cnd	Conditional
		Ind	Indicative
		Nec	Necessitative
		Opt	Optative
		Imp	Imperative
		Pot	Potential
Number	5770	Plur	Plural
		Sing	Singular

[12] `https://universaldependencies.org/u/feat/index.html`

NumType	344	Card	Cardinal number or corresponding interrogative / relative / indefinite / demonstrative word
		Ord	Ordinal number or corresponding interrogative / relative / indefinite / demonstrative word
PartType	332	Int	Particle type Interrogative
Person	3593	1	First person
		2	Second person
		3	Third person
Polarity	2062	Neg	Negative
		Pos	Positive
Polite	447	Form	Formal form
Poss	22	Yes	Possessive
PronType	330	Dem	Demonstrative pronoun, determiner, numeral or adverb
		Emp	Emphatic determiner
		Int	Interrogative pronoun, determiner, numeral or adverb
		Prs	Personal or possessive personal pronoun or determiner
PunctType	1299	Comm	comma
		Peri	Period
		Qest	Question
Reflex	3	Yes	Reflexive
Sandhi	252	C	*Sandhi* ச்
		K	*Sandhi* க்
		P	*Sandhi* ப்
		T	*Sandhi* த்
Tense	1720	Fut	Future
		Past	Past
		Pres	Present
		Fin	Finite
VerbForm	2258	Conv	Converb, transgressive, adverbial participle, verbal adverb
		Ger	Gerund
		Inf	Infinitive
		Part	Participle, verbal adjective

		Vnoun	Verbal noun, masdar
		Act	Active
Voice	146	Pass	Passive

6.8.3 Dependency relations

Table 6.11: Dependency labels used in my datasets

Dependency label	Count	Dependency label	Count
acl	69	dep	12
advcl	377	det	67
advmod	331	flat	201
amod	48	iobj	28
appos	5	mark	50
aux	227	nmod	592
case	35	nsubj	396
cc	166	nummod	64
ccomp	59	obj	374
compound	477	obl:tmod	1275
conj	8	punct	348
cop	60	root	1825
		xcomp	1542

Dependency annotations constitute the key annotation that captures the syntactic relationship among tokens. Table 6.11 shows the list of labels used in the datasets and the number of times each of them occurred in the datasets. The UD has 37 dependency labels derived from de Marneffe et al. (2014). These 37 dependency labels can be extended via the use of sub-types to capture language-specific dependency information. For instance, in my case, I have extended the nsubj to nsubj:nc (27 instances) and nsubj:pass (40 instances) to mark the non-canonical subjects and subjects of passive constructions, respectively. I have used the following 21 language-specific extended dependency annotations: acl:relcl, advcl:cond, advmod:emph, advmod:lmod, advmod:tmod, aux:neg, aux:pass, compound:lvc, compound:redup, compound:svc, nmod:poss, nsubj:nc, nsubj:pass, obl:abl, obl:agent, obl:arg, obl:cmpr, obl:inst, obl:loc, and obl:soc. Table 6.11 gives an aggregated count of these extended annotations. For instance, instead of giving nsubj:nc - 27, nsubj-40, and nsubj-329, I have given a grand total of 396 on nsubj. These extended annotations will be shown in the dependency structures discussed in the following sections.

Table 6.12: Details of Training, Development and Testing datasets

Dataset	**Corpus**	**Size - no of sentences**
Training	*Thamizhi*TB	1000
Development	*Thamizhi*TB+MWTT	275+135
Test	*Thamizhi*TB+MWTT	275+400

6.9 Training and Evaluation

As discussed in Section 6.7, the training and development datasets have been prepared using *Thamizhi*POSt, *Thamizhi*Morph, *Thamizhi*UDparser, and verified manually. The details of the prepared data are outlined in Table 6.12. This section outlines how I have trained the syntactic parser — version 2 of *Thamizhi*UDparser — with newly prepared datasets as outlined in the previous sections.

I have tried three tools and two parsing approaches in order to develop the parser. Further, I have also experimented with multi-lingual parsing along with Telugu and Hindi. As briefed in Section 6.7, I have also tried multi-lingual parsing with other languages such as Turkish and Arabic. However, I have found Hindi and Telugu to provide a relatively better scores. Therefore, for this second development stage, I simply used Hindi and Telugu languages.

I kept the initial weight initialisation the same when training models using a seed, and I tried changing hyper-parameters such as epochs, learning rate, batch size *etc.*. However, I found that the default values for most of these parameters give a better performance. These tools also have built-in support for techniques such as early stopping. Therefore, even if we set a higher epoch, the training is stopped if convergence occurs early. For instance, I set the number of epochs to 1000 when training a model using the Stanza toolkit, but it always completes the training before that, i.e. when the loss stabilises. Therefore, I mainly focused on different approaches and training-validation datasets for the parser development instead of tweaking these parameters further.

Table 6.13 outlines the parser I have trained, datasets, type of algorithms, and the score. As shown, the Stanza-based parser provides a better LAS 75.53 when testing it using the Tamil-MWTT treebank. As can be observed from the table, I used Stanza, TranKit, and the uuparser to implement the transition-based parsing models and uuparser to implement graph-based and multi-lingual training-based parsing models since other tools do not support these approaches.

Table 6.14 represents the results of experimentation with uuparser-based multi-lingual parsing. As shown, I also have applied varied approaches to see which combination works better, apart from varying languages. According to the results, the graph-based approach gives a better score.

Beyond these coarse-level evaluations, I have also tried a fine-grained evaluation to

Table 6.13: The performance of different tools and approaches

Parser implementation	**Approach**	**LAS**
Stanza-based *Thamizhi*UDparser version 2	Transition-based	**75.53**
TranKit-based Parser	Transition-based	70.70
uuparser-based Parser	Transition-based	69.76
uuparser-based Parser	Graph-based	69.07

Table 6.14: The results of multi-lingual parsing using uuparser and graph-based and transition-based algorithms

Training dataset	**Approach**	**LAS**
Tamil+Telugu	Transition-based	71.35
Tamil+Hindi	Transition-based	70.77
Tamil+Telugu+Hindi	Transition-based	70.77
Tamil+Telugu	Graph-based	**73.53**
Tamil + Hindi	Graph-based	69.55
Tamil+Telugu+Hindi	Graph-based	**73.42**

check the score of different syntactic constructions. Instead of doing this experiment with all the trained models I have tabulated, I solely made use of the best performing *Thamizhi*UDparser version 2.0, which is based on the Stanza toolkit. In the previous experiment reported in Sarveswaran and Dias (2020), a uuparser-based multi-lingual parser gave better performance. However, in this new experiment, the Stanza-based parser gives better results. Several factors may have contributed to this observation, including, the amount of data, quality of annotations, and the length of the sentences. Following that, I made use of a test set I had prepared, using sentences taken from a Tamil grammar book for this evaluation. All these sentences are classified into five categories, as shown in Table 6.15.

Table 6.15: The fine-grained evaluation results

Data-set	**LAS**
MWTT	75.53
Thamizhi+MWTT	68.98
Nominal predicated sentences	78.74
Transitive sentences	77.19
Intransitive sentences	74.37
Complex sentences	56.76
Coordinated sentences	60.56

6.10 Error Analysis

In addition to the evaluation I have shown in the previous section, in this section, I discuss how version 2 of *Thamizhi*UDparser —*Thamizhi*UDparserV2 — analyses different constructions and what could be the possible reasons for the score I got for each construction category. This analysis will give a better understanding of how the *Thamizhi*UDparserV2 performs with different types of syntactic constriction. I have additionally compared these results with the analysis of Stanza's pre-trained parser available online.[13] I have used CoNLL-U viewer,[14] a web-based viewer for documents in the CoNLL-U format with which to draw the dependency diagrams.

6.10.1 Nominal predicates

*Thamizhi*UDparserV2 gets the dependency analysis correct for the nominal predication (3), in as shown in Figure 6.2. The Stanza's pre-trained parser gets the NUM wrong and marks it as ADJ. Due to this, the dependency annotation also ends up wrongly marked as AMOD.

(3)

நான்	ஒரு	மாணவன்
nān̲	oru	māṇavan̲
I	a	student.NOM

'I am a student.'

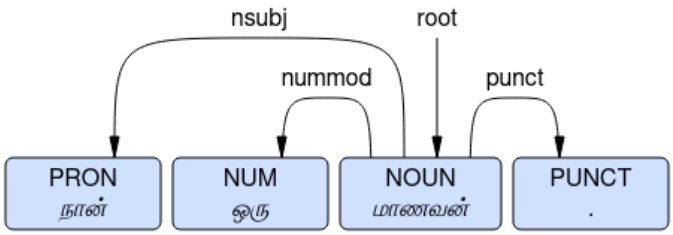

Figure 6.2: Dependency analysis of (3) using *Thamizhi*UDparserV2

[13] http://stanza.run/

[14] https://urd2.let.rug.nl/~kleiweg/conllu/

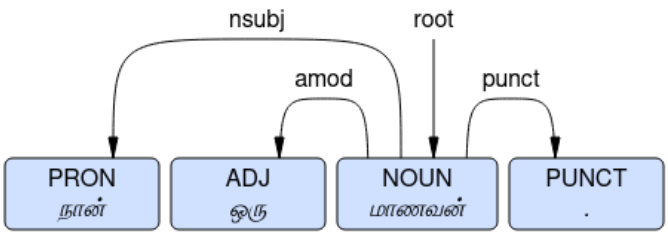

Figure 6.3: Dependency analysis of (3) using Stanza parser

However, the *Thamizhi*UDparserV2 fails when a copula is present. For instance, in (4), although the analysis gets the Head of the construction correct, the parser mistakenly identifies the NSUBJ as an OBJ. The Stanza's pre-trained parser does not even get the root correct.

(4)

கண்ணன்	ஒரு	கவிஞன்	ஆவான்
kaṇṇaṉ	oru	kaviñaṉ	āvāṉ
kannan.NOM	a	poet	be.FUT.MASC

'Kannan is a poet.'

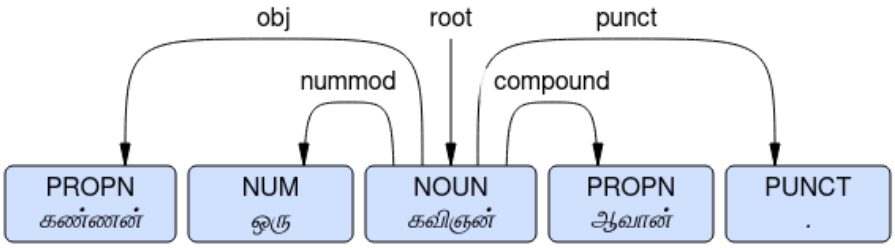

Figure 6.4: Dependency analysis of a nominal predicate shown in (4) using *Thamizhi*UDparserV2

Dative subject constructions are also included as part of the nominal predicating constructions in the categorisations. While the *Thamizhi*UDparserV2 gets most of the dative subject constructions right, and identifies the NSUBJ and the PREDICATE right, as shown in (5) and Figure 6.5. The Stanza's pre-trained parser identifies the NSUBJ wrongly as either OBL or the NMOD. This wrong annotation may well be the result of the deficiency in the Tamil-TTB treebank, in which dative subjects are not properly handled.

(5)

எனக்குப்	பசி
eṉakkup	paci
I.DAT	hungry.NOM

'I am hungry.'

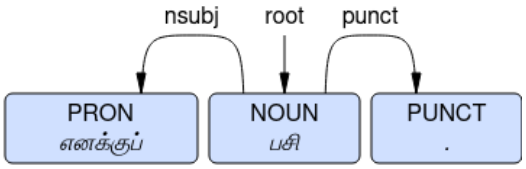

Figure 6.5: Dependency analysis of the nominal predication in (5) using *Thamizhi*UDparserV2

In general, the *Thamizhi*UDparserV2 performs better for nominal predication and dative subject constructions compared to Stanza's pre-trained parser.

6.10.2 Transitives

The *Thamizhi*UDparser gets most of the constructions in the category of transitives right, as can be observed in Table 6.15. This category also consists of constructions with dative subjects and interrogatives. The analysis of (6) is shown in Figure 6.6. Stanza's pre-trained parser fails to pass it. While it gets most parts of the analysis correct, it fails because of wrong multi-word tokenisation — it breaks the last token காணவில்லை *kāṇavillai* 'see-no' in a wrong way as காணவில் உலை, which does not make any sense, and then establish the dependency relation on top of it.

Further, in (6) the SUBJ is dropped. Since I use the UD version 2.0 specification, the subject elision is not captured as part of the annotation. Additionally, the predicate can be segmented into V+Aux, if necessary, where AUX is then used to mark negation. This is marked morphologically in my case.

(6)

எனது	பேனையைக்	காணவில்லை
e̲natu	pē̲naiyaik	kāṇavillai
I.GEN	pen.ACC	see-no

'did/do not find my pen'

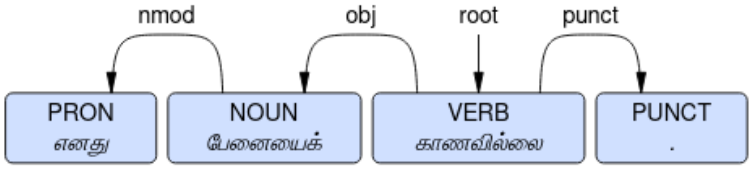

Figure 6.6: Dependency analysis of the transitive construction in (6), using the *Thamizhi*UDparserV2

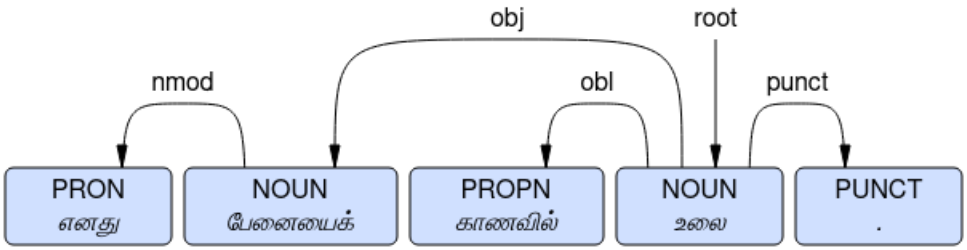

Figure 6.7: Dependency analysis of the transitive construction shown in (6), using Stanza's pre-trained parser

The *Thamizhi*UDparserV2 additionally gets the light verb constructions right, as shown in Figure 6.8. As illustrated, the construction has a compound predicate headed by a noun and a light verb which marks the functional details. The Stanza's pre-trained parser fails to get this kind of construction parsed.

(7)

குமார்	ராமனைத்	தலைவன்	ஆக்கினான்
kumār	rāmaṉait	talaivaṉ	ākkiṉāṉ
Kumar.NOM	ram.ACC	leader.NOM	make.PAST.3SGM

'Kumar made Raman the leader .'

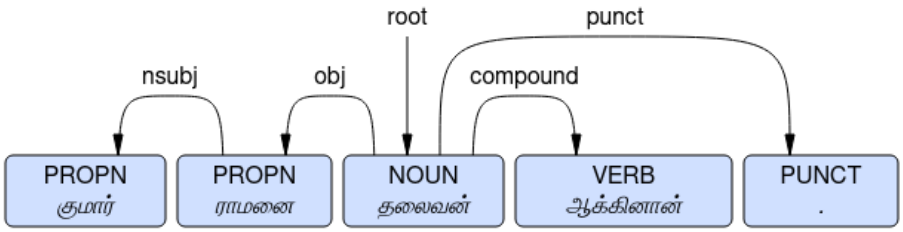

Figure 6.8: Dependency analysis of an N+V predication for (7) using *Thamizhi*UDparserV2

6.10.3 Intransitives

This construction category in the test dataset has several sub-types, including simple constructions such as subj+verb, subj+obl+verb, and dative-subj+verb. This category also includes interrogatives and passive constructions. Simple constructions are mostly handled by both Stanza's pre-trained parser and the *Thamizhi*UDparserV2.

(8)

குமார்	அப்பாவால்	அடிக்கப்பட்டான்
kumār	appāvāl	aṭikkappaṭṭāṉ
Kumar.NOM	father.INST	beat.PASSIVE.PAST.3SM

'Kumar was beaten by his father. '

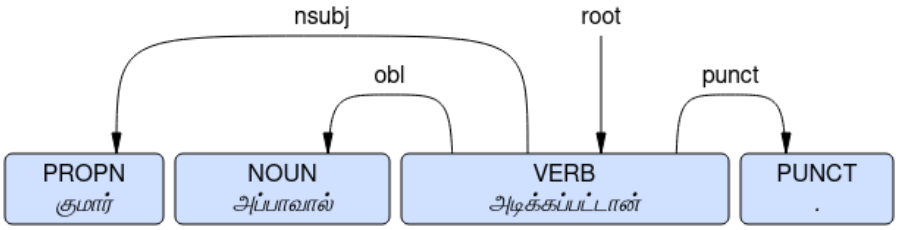

Figure 6.9: Dependency analysis of the passive construction in (8) using *Thamizhi*UDparserV2

Both analysers get the passive analysis correct, as shown in Figure 6.9. However, we can improve this analysis by marking OBL as OBL:AGNT to make the analysis more explicit.

The test set has several interrogatives, but neither *Thamizhi*UDparserV2 nor Stanza's pre-trained parser gets the interrogatives right. The training dataset mainly contains news text, and this text genre may not consist of interrogatives. Therefore, it is important to add more interrogative data to the training corpus to improve the parsing. For instance, the test set has a declarative sentence (9), and seven interrogative constructions that are derived from (9); these include three wh-questions and four different yes-no questions. However, the training dataset does not have such constructions. Therefore, the training dataset has to be improved to handle such different interrogatives and achieve a better parsing score.

(9)

அப்பா	நாளைக்கு	கொழும்புக்குப்	போகிறார்
appā	nāḷaikku	kol̲umpukkup	kol̲umpukkup
father.NOM	tommorrow.DAT	Colombo.DAT	go.PRES.3SE

'Dad is going to Colombo tomorrow '

6.10.4 Conjoined sentences

The category of conjoined sentences has conjunction and disjunction. The conjunction and disjunction can be realised morphologically in the part of speech level by involving the key part of speeches such as nouns, verbs, adjectives, and adverbs. Similar constructions can be formed at the sentence level using conjunctive and disjunctive tokens. In addition, this category also consists of other types of constructions such as conditional constructions and interrogatives. Serial verb constructions also fall under this category, in addition. The category has several different syntactical constructions, yet the training dataset does not have instances for all the construction types listed here. For instance, according to Table 6.11, the training dataset has only eight instances of CONJ relations. Because of all of these, the parser does not perform well for this category. For instance, (10) shows a sequence of actions, and the *Thamizhi*UDparserV2 gives the

analysis as shown in Figure 6.10. This analysis is wrong because the SUBJ is identified as OBJ, and the dependency is not marked correctly. The parser has, however, correctly marked the OBL relation. The proper noun கண்ணன் *Kannan* has been wrongly identified as an OBJ in most of these constructions. In the proposed analysis, each verb in the sequence of actions is considered an adverbial clause. Instead of establishing a separate relationship with the root, these relations can be merged via a sub-relation called compound:svc, as this is a serial verb.[15] I found this imbalance in the training dataset only when studying why the category of conjoined constructions showed low performance compared to other construction types.

(10)

கண்ணன்	வீட்டிற்கு	வந்து	குளித்து	சாப்பிட்டு
kaṇṇaṉ	vīṭṭiṟku	vantu	kuḷittu	cāppiṭṭu
Kannan.NOM	house.DAT	come.VPART	bath.VPART	eat.VPART

உடனே	திரும்பிச்	செல்வான்
uṭaṉē	tirumpic	celvāṉ
immediately	turn.VPART	go.FUT.3SM

'Kannan will come home, take a bath, eat and go back immediately.'

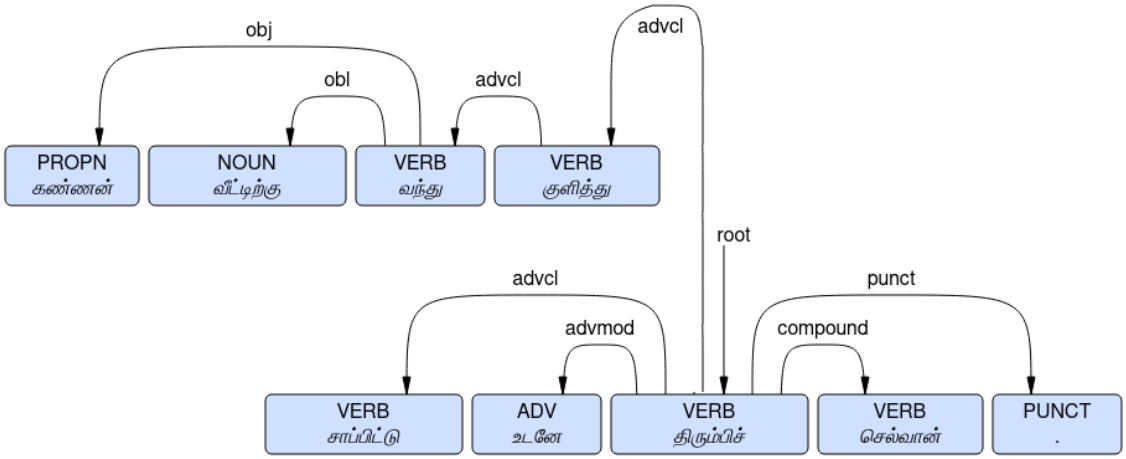

Figure 6.10: Dependency analysis of a serial verb construction in (10) using *Thamizhi*UDparserV2

The UD follows the head-initial analysis when it comes to coordination, whereby it is always the first element that is considered as the Head, and others are dependent on the Head. This is illustrated in Figure 6.11, which is an analysis of (11). In the analysis, the first VERB is related to the root, and the other conjoined phrases are related to the Head of the coordination using CONJ. As can be observed, the *-um*

[15] `https://universaldependencies.org/u/dep/compound-svc.html`

markers are separated using multi-word tokenisation and marked with CC as these function as coordination markers in this context. This construction yet again requires a more deep linguistic analysis; for instance, while the root here is 'be', and marked with present tense morphology, the whole construction is, however, considered as having happened in the 'past'. It may thus be the case that a different analysis altogether may end up being necessary.

(11)

ராதா	பாடியும்	நன்றாக	ஆடியும்	இருக்கிறாள்
rātā	pāṭiyum	nan̲rāka	āṭiyum	irukkiṟāḷ
Radha.NOM	sing.VPART-and	well.ADV	dance.VPART-and	be.PRES.2SGF

'Radha has sang and danced well.'

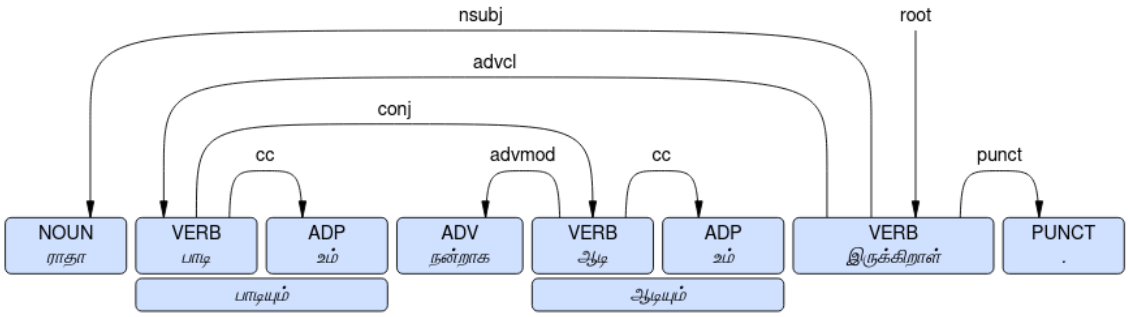

Figure 6.11: Dependency analysis of (11) using the *Thamizhi*UDparserV2

This umbrella category also includes a structure such as (12), involving the coordination of two clauses which appears to involve a gap in the first clause. Schuster et al. (2017) made a proposal to analyse gapping constructions of different types within the UD framework with the use of Extended Universal Dependencies. However, a parallel or similar analysis could not be applied to the Tamil construction in (12). In this case, one cannot just replicate the predicate to fill the gap as proposed in Schuster et al. (2017), because the predicate carries a plural marking because this has two SUBJs. Instead, I proposed an analysis as in Figure 6.12; I understand that this does not capture all linguistic information, and the dependency arcs that cross each other make this analysis a non-projective parse. Note that here *-um* is suffixed to the DAT-marked OBL nouns. It is not clear whether these *-um*s mark phrasal coordination or superiority. Further linguistic explorations are required on this type of gapping constructions and functions of *-um*.

(12)

கண்ணன்	கொழும்புக்கும்	கமால்	கண்டிக்கும்	போனார்கள்
kaṇṇaṉ	koḻumpukk-um	kamāl	kaṇṭikk-um	pōṉārkaḷ
Kannan.NOM	Colombo.DAT-and	Kamal.NOM	Kandy.DAT-and	go.PAST.3PE

'Kannan went to Colombo and Kamal to Kandy.'

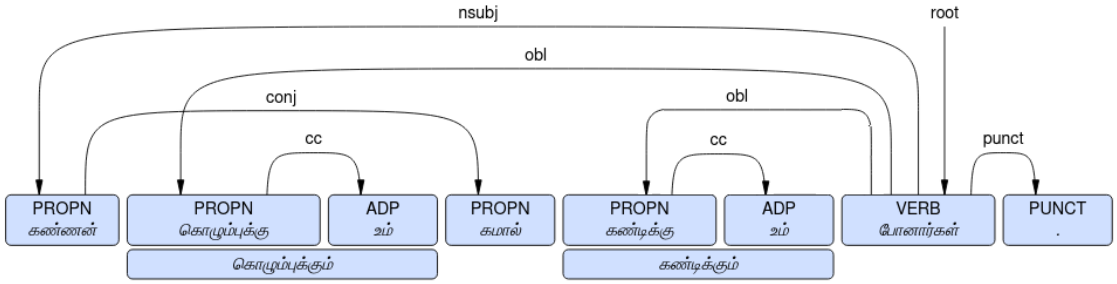

Figure 6.12: Dependency analysis of (12) using the *Thamizhi*UDparserV2

6.10.5 Complex constructions

In addition to simple sentences, the *Thamizhi*UDparserV2 is also trained with complex sentences, which include structures where more than one clause is present. Although the *Thamizhi*UDparserV2 renders a correct analysis for some sentences, the parser shows a relatively low score in comparison to other constructions. Figure 6.13 shows the analysis of (13), where the predicate of the embedded clause has a CCOMP relation with the matrix clause predicate. This embedding is marked by the complementiser, which establishes the MARK relation with the predicate of the embedded clause. The lack of training data could once again be the main reason for the score obtained; as Table 6.11 demonstrates, there are only 59 instances of complex constructions present in the training dataset.

(13)

தீர்ப்பை	மக்கள்	கூறட்டும்	என்று	அவர்	தெரிவித்தார்
tīrppai	makkaḷ	kūṟaṭṭum	eṉṟu	avar	terivittār
verdict.ACC	people	tell-may	that	he	tell.PAST.2SGE

'He told that people may say the verdict'

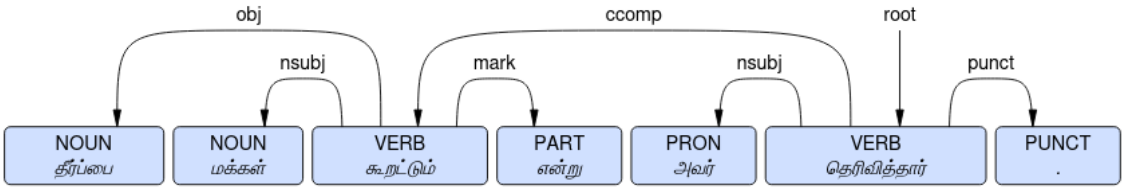

Figure 6.13: Dependency analysis of a complex construction in (13)

6.11 Conclusion

This chapter has outlined how I have developed the *Thamizhi*UDparser using the data-driven approach and the Universal Dependencies formalism. There are several off-the-shelf parsers and multi-lingual pre-trained models available (Qi et al., 2020; Kiperwasser and Goldberg, 2016; Nguyen et al., 2021) which have been trained using the Tamil-TTB treebank. I created a new dataset with more data — *Thamizhi*TB — to train, validate, and test the *Thamizhi*UDparser.

To create the required data, I adopted a hybrid approach. I first developed version 1 of the *Thamizhi*UDparser as the baseline dependency parser using multi-lingual training. In order to prepare data for this baseline system, I made use of *Thamizhi*POSt and *Thamizhi*Morph, outlined in Chapter 2 and Chapter 3, respectively. I then made use of this initial version of *Thamizhi*UDparser to annotate *Thamizhi*TB, and then validated it with two annotators and a linguist. The *Thamizhi*TB datasets consist primarily of news texts and texts from a grammar book.

Instead of developing a parser from scratch, I used the existing tools and resources to train the parser. I tried both graph-based and transaction-based parsing approaches to do the parsing. I have additionally tried these in different hyper-parameter settings to find the optimum parameters. Since the dataset I developed was small, I tried multi-lingual training along with other available languages and found that the model performs well when trained with Telugu. However, overall, the transition-based parser built using the Stanza toolkit showed better performance, as shown in Section 6.9.

Instead of just finding an overall evaluation, I have evaluated different types of syntactic constructions separately in order to be able to gauge how the parser performs on each type. As expected, when the complexity increases, the parser's performance decreases. For instance, it shows the lowest performance for complex constructions where we have two or more clauses in a sentence. The other reason for this observation could be the lack of adequate training data covering all these different categories.

All the data I have created and the parsing models are publicly available via the Web[16] for others to use and build upon. In addition, I have also incorporated this

[16] `https://github.com/sarves/thamizhi-udp`

parser into the Tamil Linguistic Information Processing toolkit[17] I have developed it so that people can use it easily in their Python environment.

The experimentation with different parsing algorithms and multi-lingual training, which are done for the first time for Tamil syntactic parsing, are also reported in this chapter. It is also observed that while the monolingual training experiment shows better performance with the transition-based algorithm, the graph-based algorithm gives better scores for multi-lingual training. Based on the evaluation, it is made clear that the *Thamizhi*UDparser, which is implemented using Stanza toolkit and transition-based algorithm, performs better when compared to other pre-trained multi-lingual parsers which handle Tamil dependency parsing. Apart from this coarse-level evaluation, I have done a fine-grain evaluation for Tamil using a categorised test set for the first time. This analysis was very useful to see when the parsers fail. This fine-grain analysis clearly shows the unbalanced nature of the test set I used. I used five umbrella syntactic categories for this purpose. However, dividing this further may be useful to identify more complex issues in syntactic parsers.

I have made the following contributions with respect to data-driven grammar development:

- compared transition-based and graph-based algorithms for the syntactic parsing of Tamil;
- for the first time, I have applied multi-lingual training for the parsing of the Tamil language;
- first time analysed how well parsers work for different syntactic constructions;
- created 1500+ sentences with the Universal Dependencies annotations and made them publicly available;
- trained and published a syntactic parser for Tamil — *Thamizhi*UDparser.

6.12 Future work

This work can be extended in at least four ways. Firstly, more data will need to be created covering different syntactic categories. The current dataset is not even, and as a result, it does not show a good performance with the complex categories. It does not, for instance, have long sentences; most of the sentences in the dataset have less than fifteen words. At the same time, significant linguistic support is also required to annotate and validate the data sets; this may be a critical factor when preparing data.

[17] `https://sarves.github.io/thamizhilip/`

Secondly, the newest version of the Universal Dependencies annotation scheme called Extended Universal Dependencies[18] (EUD) allows us to capture more syntactic information such as ellipsis, conjoined subjects/objects, relative clauses, and others. This extended annotation scheme will help to capture more deep syntactic information. One can also think of improving the language-specific dependency relations; for instance, one can mark oblique agents, serial verb constructions, light verbs *etc.* in the constructions. While these are captured in the current scheme, they are not explicitly marked using dependency sub-types. This would also improve the syntactic richness of the parsing output.

Thirdly, I have primarily used the neural-based Stanza and uuparser toolkit to implement the parsers reported in this chapter. Both of these are built on Bidirectional Long Short-Term Memory architectures. I have used the fastText pre-trained model to obtain vector representations for words before processing them. As deep neural approaches are being developed quickly, several newer models and approaches are available, such as Trankit, a transformer-based Python toolkit for multi-lingual language processing. Although I have tried TranKit, there is more that can be experimented with, indeed. Also, one can try different pre-trained models and embedding approaches. In my case, transition-based parsing worked well. However, according to the literature, graph-based parsing would have been more useful when using long sentences. This claim can also be tested for Tamil.

Fourthly, structural ambiguity is a serious problem that may have also contributed to the performance of the parser. Jurafsky and Martin (2017) have identified the following types of ambiguity in parsing related to English, namely, prepositional phrase attachment ambiguity, coordination scope ambiguity, adjectival modifier ambiguity, and verb phrase attachment ambiguity. One can invest time in exploring this direction for the Tamil language. For instance, I have come across at least the following instances of structural ambiguities. However, there may be more such ambiguities specific to the Tamil language; therefore, a detailed study on this topic is required. Such a study is important even to compile balanced datasets for training and testing.

- Subject-Object ambiguity:

குழந்தை	பந்து	விளையாடியது
kuḻantai	pantu	viḷaiyāṭiyatu
Child.NOM	ball.NOM	play.PAST.3SGN

  'Child played a ball.'

  In this construction, there are no syntactic clues to tell whether the interpretations should be such that 'child played a ball' or 'ball played a child'. Since Tamil is a free word order language, word order also does not help in this regard.

[18] `https://universaldependencies.org/u/overview/enhanced-syntax.html`

- Subject-Oblique ambiguity:

எனக்கு	பாடசாலைக்கு	போவது	பிடிக்கும்
e̲nakku	pāṭacālaikku	pōvatu	piṭikkum
I.DAT	school.DAT	go.NMLZ	like.PRES.3SGN

  'I like going to school.'

  In this construction, syntactic clues alone would not be enough to disambiguate what the SUBJ could be, since there is no subject-verb agreement. We might thus need to go beyond the mere syntax; and, further, delve into the semantic realm. This will, in turn, requires more exploration.

7 CONCLUSION

The primary objective of this study was to develop a deep syntactic parser for the Tamil language. Throughout this work, I have discussed the development of a grammar-driven parser using the Lexical-Functional Formalism and a data-driven parser — *Thamizhi*UDparser— using the Universal Dependencies formalism that captures deep syntactic information such as argumentative, attributive, and coordinative relations between the words of a given sentence. I have also discussed the development of a Part of Speech (POS) tagger — *Thamizhi*POSt, and a Morphological analyser — *Thamizhi*Morph, along with other preprocessors that were used to aid the development of the deep syntactic parsers mentioned above. I have elaborated on each of these tools in detail in the different chapters, including the necessary background and contributions.

*Thamizhi*POSt is a state-of-the-art Part of Speech tagger implemented with the use of the Deep Neural Network (DNN) based toolkit called Stanza. This tagger gives the best score of 93.27% when compared with other taggers available online. Chapter 2 outlines the complete development and the result analysis of the *Thamizhi*POSt tagger. The chapter additionally refers to a detailed account of what future work can be carried out.

*Thamizhi*Morph is a Finite-State Transducer based morphological analyser for Tamil implemented with the use of the Foma language. A paradigm-based approach has been used to capture the rich agglutinative morphology of nominal and verbal forms, in particular. Since no morphologically annotated Tamil data existed online, *Thamizhi*Morph has been used to bootstrap data for the data-driven parser and aid grammar-driven parser. *Thamizhi*Morph has been released publicly for others to make use of and build upon. The details of the implementation, along with future directions for the development of Tamil morphological analysers and generators, can be found in Chapter-3.

A grammar-driven parser based on the Lexical-Functional Grammar formalism was developed using the Xerox Linguistic Environment (XLE) toolkit. This grammar-driven parser covers common syntactic constructions derived from a Grade1 Tamil textbook and the ParGram project. Since this is implemented with the use of the ParGram specification, linguists can do a cross-lingual study along with other languages found within the ParGram project. I have made the grammar public so that others can reuse

it. In addition, I have parsed 100 sentences and published them in the Infrastructure for the Exploration of Syntax and Semantics (INESS) repository. Chapter 5 gives a detailed account of this grammar parser development, the required background theories and information to better understand the parser development, along with possible extensions to this grammar-driven parser.

*Thamizhi*UDparser is a Universal Dependencies (UD) based parser implemented using the Stanza toolkit. I annotated more than 1,500 sentences according to the UD annotation scheme to train the parser. I have also compared transition-based and graph-based parsing approaches. In order to implement the graph-based parser, I used *uuparser*. In addition to these, I also made use of multi-lingual training to train version 1 of the parser for the first time using Telugu, Hindi, and Turkish. Overall, however, the parser was developed using the Stanza toolkit, and the transition-based approach gave the current best score for parsing. In addition to this overall score, I have also tested the parser for different syntactic categories and found that its performance varies due to the complex nature of the different syntactic categories and the lack of annotated data related to certain syntactic constructions available in the training data. These details have been discussed in Chapter 6. Since the Universal Dependencies and the field around it are very active areas, several extensions to *Thamizhi*UDparser can be done; Chapter 6 also outlines some of these extensions.

Apart from all these tools and resource development, I have also spent a significant amount of time on linguistic analysis, as before writing grammar rules and training in neural networks, we should understand the data and different levels of linguistic analysis, including the morpheme level, word level, and sentence level and the contribution of each. Chapter 4 gives an account of Tamil syntax. Other chapters also provide the relevant linguistic background with which to understand and appreciate the development of the different tools and resources. This study finds that we require a proper linguistic analysis of modern Tamil in order to understand the linguistic structures and model them computationally.

I have found that although the data-driven parser is robust and handles unseen words well because of the power of the Deep neural network architecture, the grammar-driven parser gives a more detailed output to each construction. The success of the data-driven parser is, however, heavily dependent on the training data. It is for the reason that the parser does not perform well for certain categories due to the unbalance in datasets related to different syntactic constructions in the training data. Since both types of parsers are based on a formalism that supports cross-lingual analyses, these parsers are also useful for linguists and technologists to develop Natural Language Processing applications.

7.1 Software and Resources

Table 7.1: List of tools and resources have been developed in this study

Tool / Resource	Location	Description
Preprocessor	`https://github.com/sarves/thamizhi-preprocessor`	This Python script helps to validate a word according to the *naṉṉūl* (Thesikar, 1957) — a well-accepted Tamil grammar written in the 13th century. This script will process the combination of vowels and consonants in a given word as it checks whether it is acceptable, according to *naṉṉūl*. In addition, this script is also useful for doing the Unicode normalisation.
*Thamizhi*POSt	1. `http://nlp-tools.uom.lk/thamizhi-pos/` 2. `https://github.com/sarves/thamizhi-pos`	1-2. Here we can find relevant data and more information about the resource used to develop the parser. I have also published 10,000 sentences with POS annotations.
*Thamizhi*Morph	1. `http://nlp-tools.uom.lk/thamizhi-morph` 2. `https://github.com/sarves/thamizhi-morph`	1-2. Has all the resources, lexicons, and Finite-State Transducer models that I have created in order to develop the morphological analyser that is now available.

*Thamizhi*UDparser	1. `nlp-tools.uom.lk/thamizhi-udp` 2. `https://github.com/sarves/thamizhi-udp`	1-2. These two links have all the resources that I have used to train models, instances of the trained model, and how to make use of them.
ThamizhiLFGparser	1. `https://clarino.uib.no/iness/xle-web` 2. `https://git.app.uib.no/iness/pargram/-/tree/master/tamil` 3. `https://clarino.uib.no/iness/treebanks`	1. This includes the interface to parse Tamil text. 2. This link points to all the LFG rules and the lexicon I have written in order to develop the grammar-driven parser. 3. This link points to the LFG treebank. However, you need to have a username and password to access the Tamil treebank. I will find a workaround and publish the LFG treebank publicly.

Table 7.1 presents the list of tools and resources I have developed, created, and co-created. In addition, I have developed a package installer for Python (pip) called *thamizhi*lip —Thamizhi Linguistic Information Processing package with which to carry out POS tagging, Morphological analysis, and syntactic parsing. The version 0.11 of *thamizhi*lip can be found here: `https://sarves.github.io/thamizhilip`

7.2 Publications

Several publications have emerged throughout this study, jointly with my advisors and collaborators. Table 7.2 lists only those publications that are directly relevant to the tools and resources that I have developed as part of this study.

Table 7.2: List of publications that are directly related to my PhD research study

Publication	**Description**
Building a Part of Speech tagger for the Tamil Language (Sarveswaran and Dias, 2021).	This publication outlines the development of the *Thamizhi*POSt — the POS tagger.
*Thamizhi*Morph: A morphological parser for the Tamil language (Sarveswaran et al., 2021).	This journal paper discusses the process employed in the development of the morphological analyser.
ThamizhiUDp: A Dependency Parser for Tamil (Sarveswaran and Dias, 2020).	This discusses the initial data-driven parser I developed — the version 1 of *Thamizhi*UDparser.
Computational Challenges with Tamil Complex Predicates (Sarveswaran and Butt, 2019).	This paper accounts for complex predicates in Tamil and how it can be analysed within the LFG formalism.
Using Meta-Morph Rules to develop Morphological Analysers: A case study concerning Tamil (Sarveswaran et al., 2019).	This paper outlines the proposal for meta-morph rules that help making rules for the Finite-State Morphology development.
ThamizhiFST: A Morphological Analyser and Generator for Tamil Verbs (Sarveswaran et al., 2018).	This outlines how I have developed an inflectional morphological analyser for Tamil verbs using Finite-State Morphology.

REFERENCES

Abney, S. P. (1992). Parsing by Chunks. In Berwick, R., Abney, S., and Tenny, C., editors, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 257–278. Springer, Netherlands, Dordrecht.

Agesthialingom, S. (1971). A Note on Tamil Verbs. *Anthropological Linguistics*, 13(4):121–125.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In Holub, J. and Žďárek, J., editors, *Implementation and Application of Automata*, pages 11–23, Berlin, Heidelberg. Springer.

Alsina, A. (1996). *The role of argument structure in grammar: Evidence from Romance*. CSLI Publications, Stanford.

Alsina, A., Bresnan, J., and Sells, P. (1997). Complex predicates: Structure and theory. In Alsina, A., Bresnan, J., and Sells, P., editors, *Complex predicates*, pages 1–12. CSLI Publications, Stanford.

Alsina, A. and Joshi, S. (1991). Parameters in Causative Constructions. *Papers from the 27th Regional Meeting of the Chicago Linguistic Society*, 27:1–15.

Amritavalli, R. (2013). Nominal and interrogative complements in Kannada. In Lasnik, H., Park, M.-K., Miyamoto, Y., Takahashi, D., Maki, H., Ochi, M., Sugisaki, K., and Uchiboro, A., editors, *Deep insights, broad perspectives: Essays in honor of Mamoru Saito*, pages 1–21. Kaitakusha.

Annamalai, E. (2013). The Variable Relation of Verbs in Sequence in Tamil. In *NINJAL International Symposium 2013: Mysteries of Verb-Verb Complexes in Asian Languages*, pages 1–26, Japan.

Annamalai, E., Dhamotharan, A., and Ramakrishnan, A. (2014). Akarātiyiṉ putiya patippil taṟkālat tamiḻ ilakkaṇa viḷakkam (in Tamil). In Annamalai, E., Dhamotharan, A., and Ramakrishnan, A., editors, *Cre-A: Dictionary of Contemporary Tamil*, pages xxxi–xlvii. Crea-A Publishers, India.

Antony, P. (2011). Parts Of Speech Tagging for Indian Languages: A Literature Survey. *International Journal of Computer Applications*, 34:22–29.

Antony, P. and Soman, K. (2012). Computational morphology and natural language parsing for Indian languages: a literature survey. *International Journal of Scientific and Engineering Research*, 3(4):136–146.

Arden, A. H. (1910). *A Progressive Grammar of Common Tamil*. The Society for Promoting Christian Knowledge, Madras, India.

Ariaratnam, I., Weerasinghe, A., and Liyanage, C. (2014). A shallow parser for Tamil. In *International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 197–203, Colombo, Sri Lanka. IEEE.

Armengol-Estapé, J. and Costa-jussà, M. R. (2021). Semantic and syntactic information for neural machine translation. *Machine Translation*, 35(1):3–17.

Aston, G. and Burnard, L. (1997). *The BNC Handbook: Exploring the British National Corpus with SARA*. Edinburgh University Press.

Asudeh, A. (2006). Grammar Formalisms. `http://www.alta.asn.au/events/altss2004/course_notes/ALTSS-Asudeh-Grammar.pdf`, Accessed on: 20.12.2020.

Avinesh, P. and Karthik, G. (2007). Part-of-speech tagging and chunking using conditional random fields and transformation based learning. In *Shallow Parsing for South Asian Languages*, pages 21–24, Hyderabad, India.

Ballesteros, M., Bohnet, B., Mille, S., and Wanner, L. (2016). Data-driven deep-syntactic dependency parsing. *Natural Language Engineering*, 22(6):939–974.

Balusu, R. (2020). The Quotative Complementizer Says "I'm too Baroque for that". In *Formal Approaches to South Asian Languages*, pages 1–12, Kansas, USA.

Beesley, K. R. and Karttunen, L. (2003). *Finite-State Morphology: Xerox tools and techniques*. CSLI Publications, Stanford.

Bejček, E., Hajičová, E., Hajič, J., Jínová, P., Kettnerová, V., Kolářová, V., Mikulová, M., Mírovský, J., Nedoluzhko, A., Panevová, J., Poláková, L., Ševčíková, M., Štěpánek, J., and Zikánová, Š. (2013). Prague dependency treebank 3.0. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Bharati, A., Gupta, M., Yadav, V., Gali, K., and Sharma, D. M. (2009). Simple parser for Indian languages in a dependency framework. In *Proceedings of the Third Linguistic Annotation Workshop (LAW III)*, pages 162–165, Suntec, Singapore. Association for Computational Linguistics.

Bharati, A. and Sangal, R. (1993). Parsing Free Word Order Languages in the Paninian Framework. In *31st Annual Meeting of the Association for Computational Linguistics*, pages 105–111. Association for Computational Linguistics.

Bharati, A., Sangal, R., and Sharma, D. M. (2007). SSF: Shakti Standard Format guide. Available: `http://sampark.org.in/sampark/web/ssf-guide-4oct07.pdf` Accessed on: 08.09.2019.

Bhattacharyya, P., Murthy, H., Ranathunga, S., and Munasinghe, R. (2019). Indic language computing. *Communications of the ACM*, 62(11):70–75.

Bobrow, D. G., Cheslow, B., Condoravdi, C., Karttunen, L., King, T. H., Nairn, R., de Paiva, V., Price, C., and Zaenen, A. (2007). PARC's Bridge and Question Answering System. In King, T. H. and Bender, E. M., editors, *Proceedings of the Grammar Engineering Across Frameworks(GEAF07) Workshop*, Stanford. CSLI Publications.

Bögel, T., Butt, M., Hautli, A., and Sulger, S. (2007). Developing a Finite-State Morphological Analyzer for Urdu and Hindi. In Hanneforth, T. and ürzner, K.-M., editors, *Sixth International Workshop on Finite-State Methods and Natural Language Processing*, pages 86–96. Potsdam University Press.

Böhmová, A., Hajič, J., Hajičová, E., and Hladká, B. (2003). The Prague Dependency Treebank: Three-level annotation scenario. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 103–127. Springer Netherlands, Dordrecht.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Boologarambai, A. (1986). *A Study of Auxiliaries in the Old and the Middle Tamil*. PhD thesis, Centre of Advanced study in Linguistics, Annamalai University, India.

Bresnan, J. (1982). The Passive in Lexical Theory. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations*, pages 3–86. The MIT Press, Cambridge.

Britto, F. (1991). Tamil diglossia: An interpretation. *Southwest Journal of Linguistics*, 10(1):60–84.

Buchholz, S. and Marsi, E. (2006). CoNLL-X shared task on Multilingual Dependency Parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, NY, USA. Association for Computational Linguistics.

Butt, M. (1994). Machine Translation and Complex Predicates. In Trost, H., editor, *Conference on Natural Language Processing (KONVENS'94)*, pages 62–71, University of Vienna, Austria.

Butt, M. (1995). *The Structure of Complex Predicates in Urdu*. CSLI Publications, Stanford.

Butt, M. (2010). The Light Verb Jungle: Still Hacking Away. In Amberber M, H. M. and B, B., editors, *Complex Predicates in Cross-Linguistic Perspective*, pages 48–78. Cambridge University Press, Cambridge.

Butt, M., Grimm, S., and Ahmed, T. (2006). Dative subjects. Paper presented at NWO/DFG Workshop on Optimal Sentence Processing Available: `http://ling.uni-konstanz.de/pages/home/butt/nijmegen-hnd.pdf.Accessedon: 10.08.2021`.

Butt, M. and King, T. H. (2002). Urdu and the Parallel Grammar project. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization, COLING*, pages 39–45. Association for Computational Linguistics.

Butt, M. and King, T. H. (2003). Grammar writing, testing, and evaluation. In Farghaly, A., editor, *Handbook for language engineers*, number 164 in CSLI lecture notes, pages 129–180. CSLI Publications, Stanford.

Butt, M., King, T. H., and III, J. T. M. (2003). Complex Predicates via Restriction. In Butt, M. and King, T. H., editors, *Proceedings of the LFG03 Conference*, pages 92–104, Stanford. CSLI Publications.

Butt, M., King, T. H., Nino, M.-E., and Segond, F. (1999). *A Grammar Writer's Cookbook*. CSLI Publications, Stanford.

Butt, M., King, T. H., and Ramchand, G. (2008). Complex predication: Who made the child pinch the elephant? In Uyechi, L. and Wee, L. H., editors, *Reality Exploration and Discovery: Pattern Interaction in Language and Life*. CSLI Publications, Stanford.

Butt, M. and Lahiri, A. (2013). Diachronic pertinacity of light verbs. *Lingua*, 135:7–29.

Butt, M., Rajamathangi, S., and Sarveswaran, K. (2020). Mixed Categories in Tamil via Complex Categories. In Butt, M. and King, T. H., editors, *Proceedings of the LFG09 Conference*, pages 68–88, Stanford. CSLI Publications.

Caldwell, R. (1998). *A Comparative Grammar of the Dravidian or South-Indian Family of Languages*. Asian Educational Services, New Delhi, India. Original work published 1875.

Çetinoğlu, Özlem. (2009). *A Large Scale LFG Grammar for Turkish*. PhD thesis, Sabanci University, Turkey.

Chakrabarti, D., Sarma, V., and Bhattacharyya, P. (2007). Complex predicates in Indian Language Wordnets. *Lexical Resources and Evaluation Journal*, 40(3-4).

Che, W. and Zhang, Y. (2017). Deep Learning in Lexical Analysis and Parsing. In *Proceedings of the IJCNLP 2017, Tutorial Abstracts*, pages 1–2, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Chomsky, N. (2015). *Aspects of the Theory of Syntax*. MIT press.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2020). Unsupervised Cross-lingual Representation Learning at Scale. arXiv:1911.02116.

Consortium, B. (2007). British National Corpus, XML edition. From Oxford Text Archive, University of Oxford Available: `http://www.natcorp.ox.ac.uk/XMLedition/` Accessed on: 03.04.2019.

Crouch, R., Dalrymple, M., Kaplan, R. M., King, T. H., Maxwell III, J. T., and Newman, P. (2017). XLE documentation. `https://ling.sprachwiss.uni-konstanz.de/pages/xle/doc/xle_toc.html` by the Palo Alto Research Center.

Dalrymple, M. (2001). *Lexical Functional Grammar*, volume 34 of Syntax and Semantics. Academic Press, New York, NY.

Dalrymple, M., Kaplan, R. M., King, T. H., et al. (2004). Linguistic generalizations over descriptions. In Butt, M. and King, T. H., editors, *Proceedings of the LFG04 Conference*, pages 199–208, Stanford. CSLI Publications.

de Lhoneux, M., Shao, Y., Basirat, A., Kiperwasser, E., Stymne, S., Goldberg, Y., and Nivre, J. (2017a). From raw text to universal dependencies-look, no tags! In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 207–217.

de Lhoneux, M., Stymne, S., and Nivre, J. (2017b). Arc-Hybrid Non-Projective Dependency Parsing with a Static-Dynamic Oracle. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 99–104, Pisa, Italy. Association for Computational Linguistics.

de Marneffe, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., and Manning, C. D. (2014). Universal Stanford Dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, volume 14, pages 4585–4592. European Language Resources Association.

de Marneffe, M.-C., MacCartney, B., Manning, C. D., et al. (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, volume 2006, pages 449–454. European Language Resources Association.

de Marneffe, M.-C., Manning, C. D., Nivre, J., and Zeman, D. (2021). Universal Dependencies. *Computational Linguistics*, 47(2):255–308.

Devi, S. L. (2016). AUKBC Tamil Dependency-Annotated Corpus - v1. Computational Linguistics Research Group, AU-KBC Research Centre, Chennai, India.

Dhanalakshmi, V., Kumar, M. A., Shivapratap, G., Soman, K., and Rajendran, S. (2009a). Tamil POS Tagging using Linear Programming. *International Journal of Recent Trends in Engineering*, 1(02).

Dhanalakshmi, V., Padmavathy, P., Soman, K., and Rajendran, S. (2009b). Chunker for Tamil. In *International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom'09)*, pages 436–438, Kerala, India. IEEE.

Dhivya, R., Dhanalakshmi, V., Anand Kumar, M., and Soman, K. P. (2012). Clause Boundary Identification for Tamil Language Using Dependency Parsing. In Das, V. V., Ariwa, E., and Rahayu, S. B., editors, *Signal Processing and Information Technology*, pages 195–197, Berlin, Heidelberg. Springer.

Falk, Y. (2011). *Lexical-Functional Grammar*. Oxford University Press.

Fedson, V. J. (1981). *The Tamil serial or compound verb*. PhD thesis, Department of Linguistics, University of Chicago.

Floridi, L. and Chiriatti, M. (2020). GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and Machines*, 30(4):681–694.

Futrell, R., Mahowald, K., and Gibson, E. (2015). Quantifying Word Order Freedom in Dependency Corpora. In *Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015)*, pages 91–100, Uppsala, Sweden. Uppsala University, Uppsala, Sweden.

Graul, K. (1855). *Outline of Tamil grammar*. Leipzig: Dörfling Und Franke.

Grünewald, S., Oertel, F. T., and Friedrich, A. (2021). RobertNLP at the IWPT 2021 shared task: Simple enhanced UD parsing for 17 languages. In *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021)*, pages 196–203, Online. Association for Computational Linguistics.

Habash, N. (2007). Syntactic Preprocessing for Statistical Machine Translation. In *Proceedings of Machine Translation Summit XI: Papers*, pages 215–222, Copenhagen, Denmark. Association for Computational Linguistics.

Hajič, J., Panevová, J., Buráňová, E., Urešová, Z., and Bémová, A. (1999). Annotations at Analytical Level: Instructions for Annotators. Available: `https://ufal.mff.cuni.cz/pdt/Corpora/PDT_1.0/Doc/aman-en/index.html` Accessed on: 02.03.2019 ÚFAL, Praha, Czech Republic.

Hajic, J., Smrz, O., Zemánek, P., Šnaidauf, J., and Beška, E. (2004). Prague Arabic dependency treebank: Development in data and tools. In *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*, pages 110–117, Cairo, Egypt.

Hajic, J., Vidová-Hladká, B., and Pajas, P. (2001). The Prague dependency treebank: Annotation structure and support. In *Proceedings of the IRCS Workshop on Linguistic Databases*, pages 105–114, University of Pennsylvania.

Han, W., Jiang, Y., Ng, H. T., and Tu, K. (2020). A Survey of Unsupervised Dependency Parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2522–2533, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Harish, B. S. and Rangan, R. K. (2020). A comprehensive survey on Indian regional language processing. *SN Applied Sciences*, 2(7):1204.

Hart, G. L. (2000). Statement on the Status of Tamil as a Classical Language. Available: `https://southasia.berkeley.edu/tamil-classesAccessedon:10.10.2018`.

Hirschberg, J. and Manning, C. D. (2015). Advances in natural language processing. *Science*, 349(6245):261–266.

Hulden, M. (2009). Foma: a Finite-state compiler and library. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session*, pages 29–32. Association for Computational Linguistics.

Husain, S. (2009). Dependency parsers for Indian languages. In *Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing*, pages 4–6, Hyderabad, India.

Ide, N. and Suderman, K. (2004). The American National Corpus First Release. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*. European Language Resources Association.

Irākavaiyaṅkār, M. (1958). *viṉaittiripu viḷakkam (conjugation of Tamil verbs) (in Tamil)*. em. Ār. Nārāyaṇaiyaṅkār, Madurai, India.

Joseph, D. (1893). *A Grammar of Tamil*. S.P.C.K. Press, Madras, India.

Joshi, A. K. and Schabes, Y. (1997). Tree-Adjoining Grammars. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 69–123. Springer, Berlin, Heidelberg.

Jurafsky, D. and Martin, J. H. (2017). *Speech and Language Processing*. Pearson London.

Kaplan, R. M. and Bresnan, J. (1982). Lexical-Functional Grammar: A Formal System for Grammatical Representation. *Formal Issues in Lexical-Functional Grammar*, 47:29–130.

Kaplan, R. M., III, J. T. M., King, T. H., and Crouch, R. (2004). Integrating Finite-state Technology with Deep LFG Grammars. In *Proceedings of the European Summer School on Logic, Language and Information (ESSLLI) Workshop on Combining Shallow and Deep Processing for NLP*.

Kaplan, R. M. and Maxwell III, J. T. (1994). Grammar writer's workbench. *Xerox Corporation, Version 2.0*.

Kaplan, R. M. and Wedekind, J. (1993). Restriction and correspondence-based translation. In *Proceedings of the 6th Conference of the Association for Computational Linguistics European Chapter (EACL)*, pages 193–202, Utrecht University.

Karttunen, L. and Beesley, K. R. (2001). A short history of two-level morphology. ESSLLI-2001 Special Event titled 'Twenty Years of Finite-State Morphology', Available: http://www.helsinki.fi/esslli/ Accessed on: 05.09.2019.

King, T. H., Forst, M., Kuhn, J., and Butt, M. (2005). The Feature Space in Parallel Grammar Writing. *Research on Language and Computation*, 3(2):139–163.

Kiperwasser, E. and Goldberg, Y. (2016). Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Kirov, C., Sylak-Glassman, J., Que, R., and Yarowsky, D. (2016). Very-large Scale Parsing and Normalization of Wiktionary Morphological Paradigms. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, Paris, France. European Language Resources Association.

Koskenniemi, K. (1983). *Two-level morphology*. PhD thesis, University of Helsinki.

Krishnamurti, B. (2003). *The Dravidian Languages*. Cambridge University Press.

Kübler, S., McDonald, R., and Nivre, J. (2009). *Dependency Parsing*. Morgan & Claypool Publishers.

Kumar, D. and Josan, G. S. (2010). Part of Speech Taggers for Morphologically Rich Indian Languages: A Survey. *International Journal of Computer Applications*, 6:975–8887.

Kumar, M. A., Dhanalakshmi, V., and Rajendran, S. (2010a). A novel data driven algorithm for Tamil morphological generator. *International Journal of Computer Applications*, 6:52–56.

Kumar, M. A., Dhanalakshmi, V., Soman, K., and Rajendran, S. (2010b). A sequence labeling approach to morphological analyzer for Tamil language. *International Journal on Computer Science and Engineering (IJCSE)*, 2(6):1944–1951.

Kunchukuttan, A., Kakwani, D., Golla, S., C., G. N., Bhattacharyya, A., Khapra, M. M., and Kumar, P. (2020). AI4Bharat-IndicNLP Corpus: Monolingual Corpora and Word Embeddings for Indic Languages. arXiv:2005.00085.

Lehmann, T. (1993). *A Grammar of Modern Tamil*. Pondicherry Institute of Linguistics and Culture, India.

Lehmann, T. (1998). Old Tamil. In Stanford, B. S., editor, *The Dravidian languages*, pages 75–99. Routledge London, London.

Li, J., Xiong, D., Tu, Z., Zhu, M., Zhang, M., and Zhou, G. (2017). Modeling Source Syntax for Neural Machine Translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 688–697, Vancouver, Canada. Association for Computational Linguistics.

Lindén, K., Silfverberg, M., and Pirinen, T. (2009). HFST Tools for Morphology – An Efficient Open-Source Package for Construction of Morphological Analyzers. In Mahlow, C. and Piotrowski, M., editors, *State of the Art in Computational Morphology*, pages 28–47, Berlin, Heidelberg. Springer Berlin Heidelberg.

Linzen, T. and Baroni, M. (2021). Syntactic Structure from Deep Learning. *Annual Review of Linguistics*, 7:195–212.

Lisker, L. (1951). Tamil verb classification. *Journal of the American Oriental Society*, 71(2):111–114.

Lushanthan, S., Weerasinghe, A., and Herath, D. (2014). Morphological analyzer and generator for Tamil language. In *International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 190–196, Colombo, Sri Lanka. IEEE.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

Marcus, G. (2018). Deep Learning: A Critical Appraisal. arXiv:1801.00631.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

McCarthy, A. D., Silfverberg, M., Cotterell, R., Hulden, M., and Yarowsky, D. (2018). Marrying Universal Dependencies and Universal Morphology. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 91–101, Brussels, Belgium. Association for Computational Linguistics.

McCord, M. C., Murdock, J. W., and Boguraev, B. K. (2012). Deep Parsing in Watson. *IBM Journal of Research and Development*, 56(3.4):3–1.

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Menaka, S., Ram, V. S., and Devi, S. L. (2010). Morphological generator for Tamil. In *Proceedings of the Knowledge Sharing event on Morphological Analysers and Generators*, pages 82–96, LDC-IL, Mysore, India.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781.

Mohanan, K. P. (1997). Grammatical Relations and Clause Structure in Malayalam. ScholarBank@NUS Repository - Available: `http://scholarbank.nus.edu.sg/handle/10635/132853`.

Mohanan, K. P. and Mohanan, T. (1990). Dative subjects in Malayalam: Semantic information in Syntax. In Verma, M. and Mohanan, K., editors, *Experiencer Subjects in South Asian Languages*, pages 43–57. CSLI Publications, Stanford.

Mohanan, T. (1994). *Argument Structure in Hindi*. CSLI Publications, Stanford.

Mokanarangan, T., Pranavan, T., Megala, U., Nilusija, N., Dias, G., Jayasena, S., and Ranathunga, S. (2016). Tamil morphological analyzer using Support Vector Machines. In *International Conference on Applications of Natural Language to Information Systems*, pages 15–23. Springer.

Muralidaran, V. and Misra Sharma, D. (2018). Construction Grammar Based Annotation Framework for Parsing Tamil. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, pages 378–396, Cham. Springer International Publishing.

Muthuchchanmugan (2005). *Ikkala Mozhiyiyal (Modern Linguistics)*. Mahin Press, Chennai, India.

Nachinarkkiniyar (1937). *Tholkappiyam - Eluththathikaram (in Tamil)*. Thirumahal Press, Chunnakam, Sri Lanka.

Nguyen, M. V., Lai, V. D., Pouran Ben Veyseh, A., and Nguyen, T. H. (2021). Trankit: A Light-Weight Transformer-based Toolkit for Multilingual Natural Language Processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 80–90, Online. Association for Computational Linguistics.

Nivre, J. (2005). Dependency grammar and dependency parsing. MSI report, Available: `https://cl.lingfil.uu.se/~sara/kurser/5LN455-2013/lectures/5LN455-2013-12-11.pdf` Accessed on: 10.10.2020.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R. T., Petrov, S., Pyysalo, S., and Silveira, N. (2016). Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. European Language Resources Association.

Nivre, J., Hall, J., and Nilsson, J. (2006). MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association.

Nuhman, M. (1999). *aṭippaṭait tamiḻ ilakkaṇam 'Basic Tamil Grammar' (In Tamil)*. Readers' Association, Sri Lanka.

Ohta, T., Miyao, Y., Ninomiya, T., Tsuruoka, Y., Yakushiji, A., Masuda, K., Takeuchi, J., Yoshida, K., Hara, T., Kim, J.-D., et al. (2006). An intelligent search engine and GUI-based efficient MEDLINE search tool based on deep syntactic parsing. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 17–20.

Pappuswamy, U. (2005). Dative Subjects in Tamil: A Computational Analysis. *South Asian Language Review*, 43:XV.

Paramasivam, K. (2011). *ikkālat tamiḻ ilakkaṇam 'Contemporary Tamil Grammar' (in Tamil)*. Adaiyaalam, India.

Parameshwari, K. (2011). An implementation of APERTIUM morphological analyzer and generator for Tamil. *Parsing in Indian Languages*, pages 41–44.

Pattabhi, R., Rao, T., Ram, R. V. S., Vijayakrishna, R., and Sobha, L. (2007). A text chunker and hybrid POS tagger for Indian languages. In *Proceedings of International Joint Conference on Artificial Intelligence Workshop on Shallow Parsing for South Asian Languages, IIIT Hyderabad, Hyderabad, India*.

Petrov, S., Das, D., and McDonald, R. (2012). A Universal Part-of-Speech Tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2089–2096, Istanbul, Turkey. European Language Resources Association.

Pollard, C. and Sag, I. A. (1994). *Head-driven Phrase Structure Grammar*. University of Chicago Press.

Pope, G. U. (1979). *A handbook of the Tamil language*. Asian Educational Services. Originally published: 1904.

Prasain, B. (2011). *Computational Analysis of Nepali Morphology: A Model for Natural Language Processing*. PhD thesis, Faculty of Humanities and Social Sciences of Tribhuvan University, Nepal.

Premjith, B., Soman, K., and Kumar, A. M. (2018). A deep learning approach for Malayalam morphological analysis at character level. *Procedia Computer Science*, 132:47–54. International Conference on Computational Intelligence and Data Science.

Qi, P., Dozat, T., Zhang, Y., and Manning, C. D. (2018). Universal Dependency Parsing from Scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium. Association for Computational Linguistics.

Qi, P., Zhang, Y., Zhang, Y., Bolton, J., and Manning, C. D. (2020). Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108, Online. Association for Computational Linguistics.

Rahman, M. U. (2016). *Developing a Sindhi Computational Resource Grammar in Lexical Functional Grammar framework*. PhD thesis, Faculty of Engineering Science and Technology, Isra University, Hyderabad.

Rajaram, S. (1986). English-Tamil Pedagogical Dictionary. *Thanjavur: Tamil University*.

Rajendran, S. (2004). Strategies in the formation of compound nouns in Tamil. *Languages of India*, 4.

Rajendran, S. (2006). Parsing in Tamil–Present State of Art. *Language in India*, 6:8.

Rajendran, S. (2009). Preliminaries to the Preparation of a Spell And Grammar Checker for Tamil. Available: `https://www.academia.edu/12504639`, Accessed on: 03.02.2019.

Ramakrishnan, S., editor (2014). *Cre-A: Dictionary of Contemporary Tamil*. Cre-A publishers, Chennai, India.

Ramasamy, L. (2012). Prague dependency style treebank for tamil. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 1888–1894. European Language Resources Association.

Ramasamy, L. and Žabokrtský, Z. (2011). Tamil Dependency Parsing: Results Using Rule Based and Corpus Based Approaches. In Gelbukh, A. F., editor, *Computational Linguistics and Intelligent Text Processing*, pages 82–95, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ramasamy, L. and Žabokrtský, Z. (2012). Prague dependency style treebank for Tamil. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 1888–1894, Istanbul, Turkey. European Language Resources Association (ELRA).

Rehman, M. and Kazi, H. (2017). Developing a Computational Syntax of Sindhi Language in Lexical Functional Grammar Framework. *Sindh University Research Journal-SURJ (Science Series)*, 49(4):733–738.

Rosén, V., De Smedt, K., Meurer, P., and Dyvik, H. (2012). An Open Infrastructure for Advanced Treebanking. In *META-RESEARCH Workshop on Advanced Treebanking at LREC2012*, pages 22–29. European Language Resources Association.

Sarveswaran, K. and Butt, M. (2019). Computational Challenges with Tamil Complex Predicates. In Butt, M., King, T. H., and Toivonen, I., editors, *Proceedings of the LFG19 Conference, Australian National University*, pages 272–292, Stanford. CSLI Publications.

Sarveswaran, K. and Dias, G. (2020). ThamizhiUDp: A Dependency Parser for Tamil. In *Proceedings of the 17th International Conference on Natural Language Processing*, pages 200–207, Indian Institute of Technology Patna, India. NLP Association of India.

Sarveswaran, K. and Dias, G. (2021). Building a Part of Speech tagger for the Tamil Language. In *International Conference on Asian Language Processing (IALP)*, Singapore.

Sarveswaran, K., Dias, G., and Butt, M. (2018). ThamizhiFST: A Morphological Analyser and Generator for Tamil Verbs. In *2018 3rd International Conference on Information Technology Research (ICITR)*, pages 1–6. IEEE.

Sarveswaran, K., Dias, G., and Butt, M. (2019). Using Meta-Morph Rules to develop Morphological Analysers: A case study concerning Tamil. In *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing*, pages 76–86, Dresden, Germany. Association for Computational Linguistics.

Sarveswaran, K., Dias, G., and Butt, M. (2021). ThamizhiMorph: A morphological parser for the Tamil language. *Machine Translation*, 35(1):37–70.

Sarveswaran, K. and Mahesan, S. (2014). Hierarchical Tag-set for Rule-based Processing of Tamil Language. *International Journal of Multidisciplinary Studies (IJMS)*, 1(2):67–74.

Schiffman, H. F. (2004). The Tamil Case System. *South-Indian Horizons. Felicitation Volume for François Gros on the Occasion of his 70th Birthday*, pages 293–322.

Schiffman, H. F. (2008). The Ausbau issue in the Dravidian languages: the case of Tamil and the problem of purism. *International journal of the sociology of language*, 2008(191):45–63.

Schuster, S., Lamm, M., and Manning, C. D. (2017). Gapping Constructions in Universal Dependencies v2. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 123–132.

Seiss, M. (2012). A Rule-based Morphological Analyzer for Murrinh-Patha. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 751–758. Istanbul,Turkey: European Language Resources Association (ELRA), European Language Resources Association.

Seiss, M. and Nordlinger, R. (2012). An Electronic Dictionary and Translation System for Murrinh-Patha. *European Association for Computer-Assisted Language Learning (EUROCALL)*.

Selvam, M. and Natarajan, A. (2009). Improvement of rule based morphological analysis and pos tagging in Tamil language via projection and induction techniques. *International Journal of Computers*, 3(4):357–367.

Senavaraiyar (1938). *tolkāppiyam - eḻuttatikāram*. Thirumahal Press, Chunnakam, Sri Lanka.

Sha, F. and Pereira, F. (2003). Shallow Parsing with Conditional Random Fields. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 213–220. Association for Computational Linguistics.

Shanmugadas, A. (1982). *Aspects of Tamil Language and Grammar (in Tamil)*. Muttamiḻ veḷiyīṭṭuk kaḻakam, Sri Lanka.

Simons, G. F. and Fennig, C. D. (2017). Ethnologue: Languages of the world (twentieth edition). Available: `http://www.ethnologue.com` Accessed on: 30.10.2018.

Sithiraputhiran, H. (2004). *Viṉaittiripu viḷakkamum moḻiyiyal kōṭpāṭum*. International Institute of Tamil Studies.

Steedman, M. (2000). *The Syntactic Process*. MIT press Cambridge, MA.

Steever, S. B. (2005). *The Tamil Auxiliary System*. Routledge, New York.

Subbārāo, K. V. (2012). *South Asian Languages: A Syntactic Typology*. Cambridge University Press.

Sulger, S., Butt, M., King, T. H., Meurer, P., Laczkó, T., Rákosi, G., Dione, C. B., Dyvik, H., Rosén, V., and De Smedt, K. (2013). ParGramBank: The ParGram parallel treebank. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 550–560.

Sureka, K., Srinivasagan, K., and Suganthi, S. (2014). An efficiency dependency parser using hybrid approach for Tamil language. arXiv:1403.6381.

Suseendirarajah, S. (1999). *Studies in Sri Lankan Tamil Linguistics and Culture: Selected Papers of Professor Suseendirarajah*. Professor Swaminathan Suseendirarajah Sixty Fifth Birthday Commemoration Volume.

Sylak-Glassman, J. (2016). The Composition and Use of the Universal Morphological Feature Schema (UniMorph Schema). Working Draft, Center for Language and Speech Processing Johns Hopkins University, Available: `https://unimorph.github.io/doc/unimorph-schema.pdf` Accessed on: 20.10.2020.

Thavareesan, S. and Mahesan, S. (2020). Word embedding-based Part of Speech tagging in Tamil texts. In *IEEE 15th International Conference on Industrial and Information Systems (ICIIS)*, pages 478–482. IEEE.

Thesikar, S. N. (1957). *naṉṉūl viruttiyurai (In Tamil)*. Vithiyanubalana Press, Chennai, India.

Tsarfaty, R., Seddah, D., Goldberg, Y., Kübler, S., Candito, M., Foster, J., Versley, Y., Rehbein, I., and Tounsi, L. (2010). Statistical Parsing of Morphologically Rich Languages (SPMRL): what, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–12. Association for Computational Linguistics.

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 59(236):434–460.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Wedekind, J. and Orsnes, B. (2003). Restriction and Verbal Complexes in LFG: A Case Study for Danish. In Butt, M. and King, T. H., editors, *Proceedings of the LFG03 Conference*, pages 424–450, Stanford. CSLI Publications.

Zeman, D. (2008). Reusable tagset conversion using tagset drivers. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, volume 2008, pages 28–30. European Language Resources Association.

Zeman, D., Hajic, J., Popel, M., Potthast, M., Straka, M., Ginter, F., Nivre, J., and Petrov, S. (2018). CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

Zeman, D., Nivre, J., Abrams, M., Ackermann, E., Aepli, N., Aghaei, H., Agić, Ž., Ahmadi, A., Ahrenberg, L., Ajede, C. K., Aleksandravičiūtė, G., Alfina, I., Antonsen, L., Aplonova, K., Aquino, A., Aragon, C., Aranzabe, M. J., Arnardóttir, H͡., Arutie, G., Arwidarasti, J. N., Asahara, M., Ateyah, L., Atmaca, F., Attia, M., Atutxa, A., Augustinus, L., Badmaeva, E., Balasubramani, K., Ballesteros, M., Banerjee, E., Bank, S., Barbu Mititelu, V., Basmov, V., Batchelor, C., Bauer, J., Bedir, S. T., Bengoetxea, K., Berk, G., Berzak, Y., Bhat, I. A., Bhat, R. A., Biagetti, E., Bick, E., Bielinskienė, A., Bjarnadóttir, K., Blokland, R., Bobicev, V., Boizou, L., Borges Völker, E., Börstell, C., Bosco, C., Bouma, G., Bowman, S., Boyd, A., Brokaitė, K., Burchardt, A., Candito, M., Caron, B., Caron, G., Cavalcanti, T., Cebiroğlu Eryiğit, G., Cecchini, F. M., Celano, G. G. A., Čéplö, S., Cetin, S., Çetinoğlu, Ö., Chalub, F., Chi, E., Cho, Y., Choi, J., Chun, J., Cignarella, A. T., Cinková, S., Collomb, A., Çöltekin, Ç., Connor, M., Courtin, M., Davidson, E., de Marneffe, M.-C., de Paiva, V., Derin, M. O., de Souza, E., Diaz de Ilarraza, A., Dickerson, C., Dinakaramani, A., Dione, B., Dirix, P., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eckhoff, H., Eli, M., Elkahky, A., Ephrem, B., Erina, O., Erjavec, T., Etienne, A., Evelyn, W., Facundes, S., Farkas, R., Fernanda, M., Fernandez Alcalde, H., Foster, J., Freitas, C., Fujita, K., Gajdošová, K., Galbraith, D., Garcia, M., Gärdenfors, M., Garza, S., Gerardi, F. F., Gerdes, K., Ginter, F., Goenaga, I., Gojenola, K., Gökırmak, M., Goldberg, Y., Gómez Guinovart, X., González Saavedra, B., Griciūtė, B., Grioni, M., Grobol, L., Grūzītis, N., Guillaume, B., Guillot-Barbance, C., Güngör, T., Habash, N., Hafsteinsson, H., Hajič, J., Hajič jr., J., Hämäläinen, M., Hà Mỹ, L., Han, N.-R., Hanifmuti, M. Y., Hardwick, S., Harris, K., Haug, D., Heinecke, J., Hellwig, O., Hennig, F., Hladká, B., Hlaváčová, J., Hociung, F., Hohle, P., Huber, E., Hwang, J., Ikeda, T., Ingason, A. K., Ion, R., Irimia, E., Ishola, Ọ., Jelínek, T., Johannsen, A., Jónsdóttir, H., Jørgensen, F., Juutinen, M., Sarveswaran, K., Kaşıkara, H., Kaasen, A., Kabaeva, N., Kahane, S., Kanayama, H., Kanerva, J., Katz, B., Kayadelen, T., Kenney, J., Kettnerová, V., Kirchner, J., Klementieva, E., Köhn, A., Köksal, A., Kopacewicz, K., Korkiakangas, T., Kotsyba, N., Kovalevskaitė, J., Krek, S., Krishnamurthy, P., Kwak, S., Laippala, V., Lam, L., Lambertino, L., Lando, T., Larasati, S. D., Lavrentiev, A., Lee, J., Lê Hồng, P., Lenci, A., Lertpradit, S., Leung, H., Levina, M., Li, C. Y., Li, J., Li, K., Li, Y., Lim, K., Lindén, K., Ljubešić, N., Loginova, O., Luthfi, A., Luukko, M., Lyashevskaya, O., Lynn, T., Macketanz, V., Makazhanov, A., Mandl, M., Manning,

C., Manurung, R., Mărănduc, C., Mareček, D., Marheinecke, K., Martínez Alonso, H., Martins, A., Mašek, J., Matsuda, H., Matsumoto, Y., McDonald, R., McGuinness, S., Mendonça, G., Miekka, N., Mischenkova, K., Misirpashayeva, M., Missilä, A., Mititelu, C., Mitrofan, M., Miyao, Y., Mojiri Foroushani, A., Moloodi, A., Montemagni, S., More, A., Moreno Romero, L., Mori, K. S., Mori, S., Morioka, T., Moro, S., Mortensen, B., Moskalevskyi, B., Muischnek, K., Munro, R., Murawaki, Y., Müürisep, K., Nainwani, P., Nakhlé, M., Navarro Horñiacek, J. I., Nedoluzhko, A., Nešpore-Bērzkalne, G., Nguyễn Thị, L., Nguyễn Thị Minh, H., Nikaido, Y., Nikolaev, V., Nitisaroj, R., Nourian, A., Nurmi, H., Ojala, S., Ojha, A. K., Olúòkun, A., Omura, M., Onwuegbuzia, E., Osenova, P., Östling, R., Øvrelid, L., Özateş, Ş. B., Özgür, A., Öztürk Başaran, B., Partanen, N., Pascual, E., Passarotti, M., Patejuk, A., Paulino-Passos, G., Peljak-Łapińska, A., Peng, S., Perez, C.-A., Perkova, N., Perrier, G., Petrov, S., Petrova, D., Phelan, J., Piitulainen, J., Pirinen, T. A., Pitler, E., Plank, B., Poibeau, T., Ponomareva, L., Popel, M., Pretkalniņa, L., Prévost, S., Prokopidis, P., Przepiórkowski, A., Puolakainen, T., Pyysalo, S., Qi, P., Rääbis, A., Rademaker, A., Rama, T., Ramasamy, L., Ramisch, C., Rashel, F., Rasooli, M. S., Ravishankar, V., Real, L., Rebeja, P., Reddy, S., Rehm, G., Riabov, I., Rießler, M., Rimkutė, E., Rinaldi, L., Rituma, L., Rocha, L., Rögnvaldsson, E., Romanenko, M., Rosa, R., Roșca, V., Rovati, D., Rudina, O., Rueter, J., Rúnarsson, K., Sadde, S., Safari, P., Sagot, B., Sahala, A., Saleh, S., Salomoni, A., Samardžić, T., Samson, S., Sanguinetti, M., Särg, D., Saulīte, B., Sawanakunanon, Y., Scannell, K., Scarlata, S., Schneider, N., Schuster, S., Seddah, D., Seeker, W., Seraji, M., Shen, M., Shimada, A., Shirasu, H., Shohibussirri, M., Sichinava, D., Sigurðsson, E. F., Silveira, A., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Skachedubova, M., Smith, A., Soares-Bastos, I., Spadine, C., Steingrímsson, S., Stella, A., Straka, M., Strickland, E., Strnadová, J., Suhr, A., Sulestio, Y. L., Sulubacak, U., Suzuki, S., Szántó, Z., Taji, D., Takahashi, Y., Tamburini, F., Tan, M. A. C., Tanaka, T., Tella, S., Tellier, I., Thomas, G., Torga, L., Toska, M., Trosterud, T., Trukhina, A., Tsarfaty, R., Türk, U., Tyers, F., Uematsu, S., Untilov, R., Urešová, Z., Uria, L., Uszkoreit, H., Utka, A., Vajjala, S., van Niekerk, D., van Noord, G., Varga, V., Villemonte de la Clergerie, E., Vincze, V., Wakasa, A., Wallenberg, J. C., Wallin, L., Walsh, A., Wang, J. X., Washington, J. N., Wendt, M., Widmer, P., Williams, S., Wirén, M., Wittern, C., Woldemariam, T., Wong, T.-s., Wróblewska, A., Yako, M., Yamashita, K., Yamazaki, N., Yan, C., Yasuoka, K., Yavrumyan, M. M., Yu, Z., Žabokrtský, Z., Zahra, S., Zeldes, A., Zhu, H., and Zhuravleva (2020). Universal Dependencies 2.7. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Zeman, D., Popel, M., Straka, M., Hajič, J., Nivre, J., Ginter, F., Luotolahti, J.,

Pyysalo, S., Petrov, S., Potthast, M., Tyers, F., Badmaeva, E., Gokirmak, M., Nedoluzhko, A., Cinková, S., Hajič jr., J., Hlaváčová, J., Kettnerová, V., Urešová, Z., Kanerva, J., Ojala, S., Missilä, A., Manning, C. D., Schuster, S., Reddy, S., Taji, D., Habash, N., Leung, H., de Marneffe, M.-C., Sanguinetti, M., Simi, M., Kanayama, H., de Paiva, V., Droganova, K., Martínez Alonso, H., Çöltekin, Ç., Sulubacak, U., Uszkoreit, H., Macketanz, V., Burchardt, A., Harris, K., Marheinecke, K., Rehm, G., Kayadelen, T., Attia, M., Elkahky, A., Yu, Z., Pitler, E., Lertpradit, S., Mandl, M., Kirchner, J., Alcalde, H. F., Strnadová, J., Banerjee, E., Manurung, R., Stella, A., Shimada, A., Kwak, S., Mendonça, G., Lando, T., Nitisaroj, R., and Li, J. (2017). CoNLL 2017 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada. Association for Computational Linguistics.

Zhou, J., Zhang, Z., Zhao, H., and Zhang, S. (2020). LIMIT-BERT : Linguistics Informed Multi-Task BERT. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4450–4461, Online. Association for Computational Linguistics.

Zvelebil, K. (1973). *The Smile of Murugan: On Tamil Literature of South India*. Brill.