CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Analysis and Generation of Wikidata Descriptions Focusing on Bangla Language

Master's thesis in Computer science and engineering

Mohammad Rakib Imtiaz

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Master's thesis 2025

Analysis and Generation of Wikidata Descriptions Focusing on Bangla Language

Mohammad Rakib Imtiaz

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Analysis and Generation of Wikidata Descriptions Focusing on Bangla Language
Mohammad Rakib Imtiaz

Supervisor: Inari Listenmaa, Department of Computer Science and Engineering
Examiner: Aarne Ranta, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Analysis and Generation of Wikidata Descriptions Focusing on Bangla Language
Mohammad Rakib Imtiaz
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

We present a Grammatical Framework (GF)-based resource grammar for Bangla designed to automatically generate structured natural language descriptions for Wikidata entities. The system covers multiple entity types including cities, universities, islands, lakes, and humans. Unlike statistical or black-box models, our approach uses a rule-based grammar that guarantees grammatical correctness and structural consistency. Evaluations on more than 76,000 entities demonstrate high coverage (over 99%) and strong alignment with source descriptions, as shown by multilingual embedding similarity. Our results show that the generated Bangla descriptions not only complement existing entries but often exceed them in semantic consistency. This work offers a practical solution for enhancing low-resource language content in multilingual knowledge bases.

Keywords: grammatical framework, bangla, wikidata, natural language generation, computational linguistics, resource grammar library.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Inari Listenmaa, for her continuous support and guidance throughout this thesis. Her availability for questions over Discord, as well as insightful in-person meetings, were invaluable in shaping the direction and quality of this work. I am also thankful to my examiner, Professor Aarne Ranta, for his feedback and for enabling this opportunity within the Grammatical Framework research community.

Special thanks to Xiao Bokun, whose Wikidata tools were instrumental in generating Bangla descriptions and providing the data used in this project.

I am profoundly grateful to my parents for their unwavering support and encouragement during this journey. Lastly, I thank Allah for granting me the strength, perseverance, and opportunity to complete this work.

Mohammad Rakib Imtiaz, Gothenburg, 2025-09-03

Contents

List of Figures xi

List of Tables xiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Aim . . . 2
- 1.3 Grammatical Framework . . . 2
- 1.4 Grammatical Framework Example . . . 3
- 1.5 Bangla Language . . . 5
- 1.6 Bangla Writing System . . . 5
- 1.7 Related Work . . . 6

2 Morphology 9
- 2.1 Noun . . . 9
- 2.2 Pronoun . . . 12
- 2.3 Verb . . . 14
- 2.4 Adjective . . . 20
- 2.5 Adverb . . . 20
- 2.6 Postposition . . . 21
- 2.7 Conjunction . . . 22
- 2.8 Number System . . . 23
  - 2.8.1 Digit . . . 23
  - 2.8.2 Numeral . . . 24
- 2.9 Symbol . . . 26

3 Syntax 29
- 3.1 Common Noun (CN) . . . 29
- 3.2 Proper Noun (PN) . . . 29
- 3.3 Quantifiers, Determiners, and Numerals . . . 30
  - 3.3.1 Quantifiers . . . 30
  - 3.3.2 Determiners . . . 30
  - 3.3.3 Numerals (Num) . . . 31
- 3.4 Noun Phrase (NP) . . . 31
- 3.5 Constructing Noun Phrases . . . 32

3.6 Constructing Common Nouns . . . 33
3.7 Constructing Adverbial phrases (Adv) . . . 34
3.8 Constructing Adjective phrases (AP) . . . 35
3.9 Constructing Utterance-Level Noun Phrases (Utt) . . . 36

4 Description Generation 37
4.1 Lexicon Collection . . . 37
4.2 Human Descriptions . . . 37
4.3 Location Descriptions . . . 40

5 Results 47
5.1 Evaluation Criteria . . . 47
5.2 Evaluation Methods . . . 48
5.3 Evaluation Results . . . 49
5.4 Discussion . . . 50
5.5 Future Work . . . 51

6 Conclusion 53

Bibliography 55

A Appendix 1 I

List of Figures

4.1 Syntax tree of a person's description with same nationality and birthplace 39
4.2 Syntax tree of a person's description with different nationality and birthplace . . . 40
4.3 Syntax tree of a person's description with unknown nationality . . . 40
4.4 Syntax tree of a capital city description . . . 42
4.5 Syntax tree of a city description . . . 43
4.6 Syntax tree of a university description . . . 44
4.7 Syntax tree of an island description . . . 44
4.8 Syntax tree of an island description without surrounding water body information . . . 45
4.9 Syntax tree of a lake description . . . 45

List of Tables

1.1 Bangla Vowels and Consonants 6
1.2 Bangla Vowels and Their Conjunct Forms 6

2.1 Bangla Animate Noun Inflections 11
2.2 Bangla Inanimate Noun Inflections 12
2.3 Class 1 Verb Conjugation (template with `high` and `low` stems) 16
2.4 Class 2 Verb Conjugation 17
2.5 Class 3 Verb Conjugation 18
2.6 Class 4/5/6 Verb Conjugation 19
2.7 Bangla Digits 23

5.1 Coverage Comparison of Original and Generated Bangla Descriptions . . 50
5.2 Cosine Similarity Scores Between Bangla Descriptions and Their English/Chinese Counterparts 50

1

Introduction

Wikidata, a collaboratively edited multilingual website hosted by the Wikimedia Foundation, is a critical repository of structured data, encompassing items with labels, descriptions, and aliases across numerous languages. Wikidata faces significant challenges in achieving linguistic uniformity and accuracy despite its extensive reach. Descriptions of items often vary considerably between languages and, in some cases, are absent. These inconsistencies lead to discrepancies in meaning, redundant efforts, and confusion among users.

Addressing these challenges is crucial for advancing our understanding of how structured data can be transformed into coherent, human-readable text across diverse linguistic contexts. Strengthening the link between language-independent structured data and language-specific realizations supports the development of robust, language-agnostic content generation models, promotes linguistic equity, and fosters richer global participation in the knowledge ecosystem.

This thesis focuses on developing a systematic approach to generating Bangla descriptions for Wikidata items, ensuring that the same core information can be expressed consistently and accurately in Bangla. By leveraging the Grammatical Framework (GF), a powerful tool for defining multilingual grammar, we aim to create a GF resource tailored for Bangla. This resource will include grammatical rules and lexicons necessary for automatically generating natural-sounding Bangla descriptions of Wikidata entities. Our work will streamline Wikipedia content production, improve consistency and accuracy across different languages, and facilitate the integration of machine-generated texts into educational, cultural, and informational resources.

The broader implications of this research extend beyond Bangla. Language modeling and grammar formalization advancements can be applied to other less-resourced languages, supporting the growth and maintenance of genuinely multilingual and inclusive knowledge platforms. By addressing the technical challenges of generating consistent and reliable Bangla descriptions, this thesis contributes to the ongoing efforts to enhance the linguistic inclusivity and accuracy of Wikidata's multilingual content.

1.1 Motivation

Wikidata[1] is a collaboratively edited multilingual knowledge graph hosted by the Wikimedia Foundation. The Wikidata repository consists mainly of items with labels, descrip-

tions, and aliases. For example, the city of Gothenburg has the following descriptions on the site,

> second-largest city in Sweden and capital of the Västra Götaland County
>
> *-English*

> tätort i Göteborgs kommun, Sverige
>
> *-Swedish*

We can see that the two descriptions do not present the same amount of information. The description is also missing in many languages, such as Bangla, Japanese, and Hindi. Ideally, all languages will have a description for Gothenburg and express the same information. Wikidata also has 115,329,418 items, and volunteers cannot label all of them.

1.2 Aim

We aim to address the technical challenge of generating consistent and reliable descriptions for Wikidata items by leveraging the Grammatical Framework (GF) and Wikifunctions. GF allows multilingual grammar abstracting from specific language details, bridging language-independent representations and language-specific realizations. Wikifunction is a function we can run on Wikidata items. We plan to create wikifunctions that will automate description generation in a language-agnostic form in GF. We also plan to create a GF resource tailored for Bangla, including both grammatical rules and lexicons and then use this framework to generate natural-sounding Bangla descriptions of Wikidata entities automatically. To ensure the effectiveness of our approach, we will evaluate the generated texts. This process will improve the consistency, maintainability, and linguistic inclusivity of Wikidata's multilingual content.

1.3 Grammatical Framework

Grammatical Framework (GF) has been established as a powerful tool for defining multilingual grammars, offering a language-independent abstract syntax and multiple parallel concrete syntaxes.[2][3] It provides a robust platform for defining grammars in a language-independent manner.

The Resource Grammar Library (RGL) is a comprehensive set of natural language grammars implemented in GF designed to support multilingual applications.[4] The RGL provides a familiar abstract syntax that serves as a language-independent representation of syntactic structures, which can be mapped to language-specific concrete syntaxes. This allows for generating grammatically correct and natural-sounding text in multiple languages. The library includes many syntactic and morphological rules, covering essential grammatical constructs such as noun phrases, verb conjugations, and sentence formations. By leveraging the RGL, developers can create multilingual applications with consistent and accurate language generation, reducing the need for extensive linguistic expertise.

Grammatical Framework (GF) defines the necessary grammatical rules and lexicons for

generating natural-sounding Bangla descriptions of Wikidata entities. By analyzing the Wikidata dataset, we identify the key grammatical structures and terms required for description generation in Bangla. Using RGL's abstract syntax, we create a concrete syntax specific to Bangla. This concrete syntax includes rules for noun phrases, verb conjugations, and sentence constructions that are typical in Bangla. Additionally, we develop lexicons for common Wikidata entities, ensuring that the generated descriptions are both accurate and linguistically appropriate. This approach allow us to systematically generate consistent and reliable Bangla descriptions, enhancing the linguistic inclusivity and accuracy of Wikidata's multilingual content.

1.4 Grammatical Framework Example

To illustrate how GF works, consider the following example of a simple food grammar. This example demonstrates the abstract syntax and how it can be mapped to concrete syntaxes in different languages.

```
abstract Foods = {
flags startcat = Comment ;
cat
    Comment ; Item ; Kind ; Quality ;
fun
    Pred : Item -> Quality -> Comment ;
    This, That, These, Those : Kind -> Item ;
    Mod : Quality -> Kind -> Kind ;
    Wine, Cheese, Fish, Pizza : Kind ;
    Very : Quality -> Quality ;
    Fresh, Warm, Italian,
      Expensive, Delicious, Boring : Quality ;
}
```

In this abstract syntax:

- Categories (`cat`): Define the types of objects in the grammar, such as `Comment`, `Item`, `Kind`, and `Quality`.
- Functions (`fun`): Describe the relationships and operations between these categories. For example, the `Pred` function takes an `Item` and a `Quality` to output a `Comment`.

The `Mod` function, which modifies a `Kind` with a `Quality`, can be used to construct phrases like *Fresh Pizza*.

$$Mod(Fresh : Quality, Pizza : Kind) \rightarrow Fresh \quad Pizza : Kind$$

The concrete syntax defines how the abstract syntax is realized in a specific language. For instance, in English, the abstract syntax above might be mapped to the following concrete syntax:

```
concrete FoodsEng of Foods = {
  flags language = en_US;
```

```
    lincat
      Comment, Quality = {s : Str} ;
      Kind = {s : Number => Str} ;
      Item = {s : Str ; n : Number} ;
    lin
      Pred item quality =
        {s = item.s ++ copula ! item.n ++ quality.s} ;
      This  = det Sg "this" ;
      That  = det Sg "that" ;
      These = det Pl "these" ;
      Those = det Pl "those" ;
      Mod quality kind =
        {s = \\n => quality.s ++ kind.s ! n} ;
      Wine = regNoun "wine" ;
      Cheese = regNoun "cheese" ;
      Fish = noun "fish" "fish" ;
      Pizza = regNoun "pizza" ;
      Very a = {s = "very" ++ a.s} ;
      Fresh = adj "fresh" ;
      Warm = adj "warm" ;
      Italian = adj "Italian" ;
      Expensive = adj "expensive" ;
      Delicious = adj "delicious" ;
      Boring = adj "boring" ;
    param
      Number = Sg | Pl ;
    oper
      det : Number -> Str ->
        {s : Number => Str} -> {s : Str ; n : Number} =
          \n,det,noun -> {s = det ++ noun.s ! n ; n = n} ;
      noun : Str -> Str -> {s : Number => Str} =
        \man,men -> {s = table {Sg => man ; Pl => men}} ;
      regNoun : Str -> {s : Number => Str} =
        \car -> noun car (car + "s") ;
      adj : Str -> {s : Str} =
        \cold -> {s = cold} ;
      copula : Number => Str =
        table {Sg => "is" ; Pl => "are"} ;
  }
```

We provide a detailed explanation of the `FoodsEng` concrete syntax for English, which maps the abstract syntax of the `Foods` grammar to English-specific structures.

- Categories (`lincat`): These define how abstract categories are represented in the concrete syntax. Each category is mapped to a record or table that describes its linguistic properties.
    - `Comment` and `Quality` are mapped to records with a single field `s` of type `Str`

(string).

- `Kind` is mapped to a table that depends on `Number` (singular or plural), indicating that nouns may have different forms depending on their number.
- `Item` is mapped to a record with two fields: `s` (the string representation of the item) and `n` (the number of the item, either singular or plural).

- Functions (`lin`): These define how abstract functions are realized in the concrete syntax. For example, the `Pred` function constructs a sentence by concatenating the item, a verb ("is"/"are"), and the quality.
- Parameters (`param`): These define language-specific grammatical features. In this case, `Number` is a parameter with two values: `Sg` (singular) and `Pl` (plural).
- Operations (`oper`): These are helper functions used to simplify the definition of rules. Determiners like This, That, These, and Those are defined using a helper function `det`, which handles number agreement. Adjectives and nouns are defined using helper functions like `adj`, `noun`, and `regNoun`.

1.5 Bangla Language

Bengali, or Bangla (বাংলা), is an Indo-Aryan language with a deep historical and literary tradition, serving as the national and official language of Bangladesh and the second most widely spoken language in India. It is the native language of over 242 million people, making it the seventh most spoken language in the world by native speakers. Linguistically, Bangla derives from Indo-Aryan languages like Magadhi Prakrit and Pali.

Bangla holds a significant place in socio-political history, particularly through the Bengali Language Movement of 1952, which established the right to linguistic identity and contributed to the eventual emergence of Bangladesh as a sovereign nation. This struggle also led to UNESCO's recognition of February 21 as International Mother Language Day. The language encompasses a rich literary corpus, including contributions from Nobel Laureate Rabindranath Tagore, and continues to be a central medium for cultural, academic, and intellectual discourse in South Asia and beyond.

1.6 Bangla Writing System

The Bangla writing system uses the Eastern Nagari script, which is related to but distinct from the Devanagari script used in Hindi, Nepali, and Sanskrit. The script is written from left to right and consists of 11 vowels and 39 consonants. (Table 1.1)

It features a distinctive horizontal line, called the "matra," running along the top of the letters, connecting them in words. Bangla follows a syllabic structure where each consonant inherently includes a vowel, forming a syllabic unit. This system accounts for the presence of two symbols for each vowel: a full vowel, which stands alone as a syllable, and a vowel sign, which attaches to consonants. (Table 1.2)

Vowels	অ(o), আ(a), ই(i), ঈ(ī), উ(u), ঊ(ū), ঋ(ṛ), এ(e), ঐ(oi), ও(o), ঔ(ou)
Consonants	ক(kô), খ(khô), গ(gô), ঘ(ghô), ঙ(ṅô), চ(cô), ছ(chô), জ(jô), ঝ(jhô), ঞ(ñô), ট(ṭô), ঠ(ṭhô), ড(ḍô), ঢ(ḍhô), ণ(ṇô), ত(tô), থ(thô), দ(dô), ধ(dhô), ন(nô), প(pô), ফ(phô/f), ব(bô), ভ(bhô), ম(mô), য(yô), র(rô), ল(lô), শ(śhô), ষ(ṣô), স(sô), হ(hô), ড়(ṛô), ঢ়(ṛhô), য়(yô), ৎ(t), ং(ṃ), ঃ(ḥ), ঁ (m)

Table 1.1: Bangla Vowels and Consonants

Vowels	অ	আ	ই	ঈ	উ	ঊ	ঋ	এ	ঐ	ও	ঔ
Conjunct Forms		া	ি	ী	ু	ূ	ৃ	ে	ৈ	ো	ৌ
Consonant with Conjunct Forms	ক (kô)	কা (ka)	কি (ki)	কী (kī)	কু (ku)	কূ (kū)	কৃ (kri)	কে (ke)	কৈ (koi)	কো (ko)	কৌ (kou)

Table 1.2: Bangla Vowels and Their Conjunct Forms

If no vowel is explicitly attached, the default vowel — the inherent vowel (অ) — is automatically used. A consonant without the inherent vowel attached is marked using (্) sign. There are also complex conjunct characters that are formed by combining multiple consonants. Instead of the symbol ., the symbol । is used as full stop marking the end of a sentence.

1.7 Related Work

The Grammatical Framework (GF) [2] has been widely used for multilingual grammar development and natural language generation (NLG). GF provides a formalism based on a logical framework that enables the construction of multilingual grammars through abstract and concrete syntaxes, ensuring both syntactic correctness and semantic coherence. Its strong typing system, based on dependent types, facilitates language processing tasks such as translation and interactive editing of syntactic trees. The concept of embedded controlled natural languages (CNLs) introduced in [5] builds upon GF's capabilities.

A foundational resource supporting GF-based development is the GF Resource Grammar Library (RGL) [4], which implements parallel grammars for over a dozen languages. The RGL enables applications in translation, localization, spoken dialogue systems, and multilingual generation by providing compositional mappings from a shared abstract syntax to language-specific constructs. This resource [3] further explores how to program with multilingual grammars.

The potential of GF for large-scale multilingual NLG has been further explored here [6].

It [6] discusses strategies for generating multilingual content from abstract representations, emphasizing modularity and code reuse across languages. Similarly, the generation of Wikipedia articles from Wikidata using GF and WordNet resources has been demonstrated by this paper [7], highlighting the feasibility of scalable, high-quality multilingual NLG systems.

On the computational linguistics front, this paper [8] presents the development of GF-based resource grammars for six Indo-Iranian languages. This work showcases the adaptability of GF in supporting under-resourced languages through functor-style grammar reuse and the construction of morphological lexicons using existing lexical resources.

The present work extends the Grammatical Framework to Bangla, with the aim of supporting natural language generation of entity descriptions from Wikidata. It draws inspiration from previous efforts in generating multilingual Wikipedia content using GF and WordNet [7], and follows similar methodologies to the resource grammar development in this paper [8]. The Bangla grammar rules, morphology, and syntax in this paper were sourced from this book [9].

2

Morphology

This chapter covers the morphology of Bangla grammar and their implementation in grammatical framework.

2.1 Noun

A noun is defined as a word that names a living being, object, place, feeling, or concept. They function as the subject or object of a sentence, helping to identify or describe the entities involved in an action or state. A noun in Bangla, much like in other languages, is a word that refers to a person, place, thing, quality, or idea. Examples: মানুষ –human, বই – book, স্কুল – school, প্রেম – love

In Bangla, a bare noun refers to a noun that appears without any classifiers, modifiers, or determiners (e.g., possessives, deictics, or quantifiers). Bangla follows a ”need-to-know” principle, meaning distinctions like number or definiteness are only marked when context demands it. If the meaning is already clear, classifiers and markers can be omitted.

Bangla also does not have grammatical gender, unlike many languages such as Hindi or French. Nouns in Bangla are not inherently marked as masculine or feminine, and adjectives or verbs do not change form based on the gender of the noun. For example, the same word ডাক্তার(doctor) can refer to both male and female individuals, and verbs remain the same regardless of gender. While natural gender is understood from context or can be specified with separate words (e.g., ছেলে for ”boy” and মেয়ে for ”girl”), it does not affect the grammatical structure of the sentence. Animacy of a noun however, affects number agreement and case usage. It will be explored later on.

In Bangla, nouns can be changed by several grammatical categories:

1. Number Bangla nouns inflect for number, distinguishing between:
    - Singular (Sg): ছেলে — boy
    - Plural (Pl): ছেলেরা — boys
2. Case Bangla employs case markers to denote syntactic roles:
    - Nominative (Nom): This is the default case form used for the subject of a sentence. — ছেলে স্কুলে যায়।

- Objective (Obj): This case is used to mark the direct or indirect object of a verb. — আমি ভাত দেখি।
- Genitive (Gen): This case expresses possession or close association. — মেয়েটার জামা নীল।
- Locative (Loc): This case marks location or direction — আমি বাসে ঢাকায় যাই।

3. Definiteness and Articles Although Bangla doesn't have standalone articles like "the" or "a" in English, definiteness is expressed by adding a classifier after a noun or through deictic and/or possessive adjectives:
   - Indefinite: একটা ছেলে — a boy
   - Definite: ছেলেটা — the boy, এই ছেলেটা — this boy
4. Animacy Animacy is a Grammatical feature that expresses how alive / sentient a noun is. All nouns denoting people are animate and nouns denoting animals, objects, concepts, places and others are inanimate. This feature influences number agreement and sometimes case usage.
   - Animate: মানুষ (human), ছেলে (boy)
   - Inanimate: বই (book), চেয়ার (chair)

Noun defined in GF syntax,

```
param
  Case = Nom | Gen | Obj | Loc ;
  Number = Sg
         | Pl ;

  Article = Indefinite | Definite ;
  Animacy = Inanimate | Animate ;
  NForm = Bare | Inflection Number Article Case ;
oper
  LinN : Type = {
    s :
        NForm =>
        Str ;
    numSuff : Str
    } ;

  mkLinN : Animacy -> Str -> LinN ;

  mkLinN animacy str = case animacy of {
        Animate => genLocSameN str ;
        Inanimate => nomObjSameN str
  } ;
```

The GF (Grammatical Framework) code defines Bangla noun inflection by classifying nouns based on animacy and systematically mapping them to surface forms using a com-

bination of grammatical features: number (singular/plural), definiteness (definite/indefinite), and case (nominative, objective, genitive, locative). The `mkLinN` function distinguishes between animate and inanimate nouns, routing each to a corresponding helper function—`genLocSameN` for animate and `nomObjSameN` for inanimate—that builds a table of inflected forms. These tables are implemented using GF's table construct and match grammatical features to their realizations in Bangla script. For instance, an animate singular definite noun in the nominative case is formed by appending "টা" to the noun stem (e.g., ছেলে -> ছেলেটা), while its plural definite genitive form becomes (e.g., "গুলোর": ছেলে -> ছেলেগুলোর). These inflection patterns are laid out clearly in table 2.1 and 2.2. They reflect how the code handles morphological composition in a structured, linguistically grounded way, accounting for phonological endings of stems when choosing the appropriate suffix (such as "র", "য়ের", or "ের"). Depending on animacy and other factors the suffix that is added to a number in front of a noun changes. So, there is a `numStuff string` field to store the suffix in the noun. It is usually "জন" for animate nouns and "টি" for inanimate nouns. This design demonstrates both the regularity and contextual flexibility of Bangla noun phrase formation.

Number	Definiteness	Case	Form with stem s
Singular	Indefinite	Nom	একজন s
Singular	Indefinite	Obj	একজন s + কে
Singular	Indefinite	Gen/Loc	s + র / য়ের / ের
Singular	Definite	Nom	s + টা
Singular	Definite	Obj	s + টাকে
Singular	Definite	Gen/Loc	s + টার
Plural	Indefinite	Nom	s + রা / য়েরা / েরা
Plural	Indefinite	Obj	s + দেরকে
Plural	Indefinite	Gen/Loc	s + দের
Plural	Definite	Nom	s + গুলো
Plural	Definite	Obj	s + গুলোকে
Plural	Definite	Gen/Loc	s + গুলোর

Table 2.1: Bangla Animate Noun Inflections

Number	Definiteness	Case	Form with stem s
Singular	Indefinite	Nom/Obj	একটা s
Singular	Indefinite	Gen	একটা s + র / য়ের / ের
Singular	Indefinite	Loc	একটা s + ে
Singular	Definite	Nom/Obj	s + টা
Singular	Definite	Gen	s + টার
Singular	Definite	Loc	s + টায়
Plural	Indefinite	Nom/Obj	s
Plural	Indefinite	Gen	s + র / য়ের / ের
Plural	Indefinite	Loc	s + ে
Plural	Definite	Nom/Obj	s + গুলো
Plural	Definite	Gen	s + গুলোর
Plural	Definite	Loc	s + গুলোতে

Table 2.2: Bangla Inanimate Noun Inflections

2.2 Pronoun

Pronouns play a crucial role in any language, serving as substitutes for nouns to avoid repetition and maintain fluency in speech and writing. The system of Bangla pronouns is rich and nuanced, reflecting number, familiarity, and formality.

A defining characteristic of Bangla pronouns is formality and social hierarchy. There are also degrees of politeness in 2nd and 3rd person. In 2nd person, there are three degrees of honorific: familiar, ordinary and polite. There are two degrees of politeness for the 3rd person: ordinary and polite. Another unique feature is the lack of gender distinction in third-person pronouns. Words like “সে” and “তারা” can mean “he” or “she,” “they (male)” or “they (female),” depending on the context. The pronouns also inflect in case.

Pronoun defined in GF syntax,

```
param
```

```
  Person = P1 | P2H0 | P2H1 | P2H2 | P3H1 | P3H2 ;

oper
  LinPron : Type = {
    s : Case => Str ;
    n : Number ;
    p : Person ;
    } ;

  mkPron : (i, me, my, mine : Str) -> Person -> Number -> LinPron
   = \i, me, my, mine, per, num -> {
    s = table {
      Nom => i;
      Gen => me;
      Obj => my;
      Loc => mine
    } ;
    p = per ;
    n = num
    } ;
```

The given GF (Grammatical Framework) code defines a basic structure for handling pronouns in a language by modeling their grammatical features. The `Person` type enumerates six possible grammatical persons, combining speaker, addressee, and levels of formality (e.g., `P2H0`, `P2H1` for second person with honorific levels). The `LinPron` record type represents the linearization (surface form) of a pronoun, mapping grammatical case (`Case`) to a string (`Str`) while also storing person (`p`) and number (`n`). The function `mkPron` serves as a constructor to generate a `LinPron` by taking four string forms of a pronoun (e.g., nominative, genitive, objective, locative), a `Person`, and a `Number`, and returns the structured record. The use of a table for `s` allows case-dependent forms of the pronoun to be easily retrieved. Here is an example of a bangla pronoun created in GF using the `mkPron` function,

`youPolSg_Pron = mkPron` ”আপনি” ”আপনার” ”আপনাকে” ”আপনার” `P2H2 Sg;`

This line defines a polite form of ”you” in Bangla (আপনি), appropriate when addressing someone respectfully—such as an elder, teacher, or stranger.

In conclusion, the Bangla pronoun system illustrates the language’s deep sensitivity to social context, respect, and interpersonal relationships. By encoding distinctions of formality, familiarity, and number directly into pronouns, Bangla allows speakers to navigate social hierarchies with linguistic precision. The GF syntax provides a structured and flexible way to model these nuances computationally, enabling accurate representation and generation of Bangla pronouns based on case, number, and politeness level.

2.3 Verb

Verbs are essential components of any language, serving as the core of sentences by expressing actions, events, or states of being. They indicate what the subject is doing or experiencing and often carry important grammatical information such as tense, aspect, mood, and agreement with the subject in terms of person and number.

In Bangla, verbs are the core elements that express action, state of being, or change, and they serve as the foundation around which sentences are structured. While nouns provide the content, it is verbs that supply the dynamics—indicating what is happening, who is involved, and when the action takes place. Each verb consists of a stem, which holds the basic meaning, and a verb ending, which conveys grammatical information such as person, tense, and mood.

A distinctive feature of Bangla verbs is their systematic vowel mutation, a phenomenon where the vowel in the stem changes depending on tense and context. Unlike English, where such changes often result in irregular verbs (e.g., sing → sang, come → came), Bangla applies vowel mutations in a consistent and rule-based manner, making the entire verb system remarkably regular. Verbs in Bangla are divided into six conjugation classes, primarily determined by how their stems are formed. To analyze and generate verb forms accurately, linguists identify a high stem and a low stem for each verb, which are then used to derive various conjugated forms. The class 4, 5 and 6 verbs differ only in the vowel mutation of their stems. The conjugation table for class 4, 5 and 6 verbs are the same. A bangla verb can be classified into it's conjugation class by checking the composition of it's stem word.

- Class 1 - The stem word follows consonant - vowel - consonant pattern or vowel - consonant pattern. Ex- লেখ, ওঠ
- Class 2 - The stem word follows consonant - specific vowel () - consonant pattern or specific vowel () - consonant pattern. Ex- থাক, আস
- Class 3 - The stem word follows consonant - vowel pattern. Ex-শো, হ
- Class 4 - The stem word follows consonant - specific vowel () pattern. Ex-পা, খা
- Class 5 - The stem word follows consonant - vowel consonant - vowel pattern. Ex-চালা, ঘুমা
- Class 6 - The stem word ends with extended o sound. Ex- এগ (ego), বেরো (bero)

The vowel mutations changing pronunciation do not always make a change in spelling. যাওয়া (go)verb is the only bangla verb with irregularities in it's vowel mutation. Besides a low steam (যা) and a high stem (গে) , it also has another stem for past perfect tense called গিয়ে.

Bangla recognizes eight tenses—including simple present, present continuous, present perfect, future, simple past, past continuous, past perfect, and past habitual. These verb forms are created by combining stem variations with a set of verb endings that correspond to six distinct grammatical persons: first person, three forms of second person

(familiar, ordinary and polite), and two forms of third person (ordinary and polite). However, since the verb endings for the second person polite and the third person polite are identical, they effectively share the same conjugational pattern. As a result, while Bangla distinguishes six grammatical persons in usage, there are practically only five unique verb endings used in conjugation for each tense.

Bangla verbs do not inflect for gender or number and the same endings are used for both singular and plural subjects. This streamlined system allows for consistent and relatively predictable verb conjugation once the underlying patterns of verb formation are understood. The tables 2.3, 2.4, 2.5, 2.6 refer to the conjugation pattern for Bangla verbs.

Verb defined in GF syntax,

```
param
  VForm = VF Tense Person ;
  Tense = PreSim | PreCon | PrePer | PaSim | PaCon | PaPer | PaHab
        | FuSim ;

oper
  LinV : Type = {
    s :
        Tense =>
        Person =>
        Str ;
    } ;

  mkVerb : Str -> Str -> LinV ;
```

The GF code provides a structured framework for modeling Bangla verb morphology by organizing verb inflection around grammatical categories such as tense and person. The `VForm` parameter combines `Tense` and `Person` into a single composite, capturing the full range of conjugation possibilities. At the core of this system is the `mkVerb` function, which takes two stems—the low stem and the high stem—as input and uses pattern matching to determine the verb's conjugation class based on the structure of the stem. Once classified, the function delegates to the appropriate class-specific constructor to systematically generate all verb forms across the eight tenses and five unique person categories following the conjugation tables 2.3, 2.4, 2.5, 2.6. Since both the low and high stems are provided explicitly, there is no need to compute vowel mutations dynamically, simplifying the process. This design enables efficient and consistent generation of Bangla verb forms.

```
rise_Verb = mkVerb "ওঠ্" "উঠ্";
```

This line defines the rise verb in bangla. "ওঠ্"is the low stem and "উঠ্"is the high stem. Taking both stems as input `mkVerb` creates the table for all forms of the rise verb.

Tense	1st Person	2nd Person (Familiar)	2nd Person (Ordinary)	3rd Person (Ordinary)	2nd/3rd Person (Polite)
Present Simple	high + ি	low	high + িস	low + ে	low + েন
Present Continuous	high + ছি	high + ছো	high + ছিস	high + ছে	high + ছেন
Present Perfect	high + েছি	high + েছো	high + েছিস	high + েছে	high + েছেন
Past Simple	high + লাম	high + লে	high + লি	high + লো	high + লেন
Past Continuous	high + ছিলাম	high + ছিলে	high + ছিলি	high + ছিলো	high + ছিলেন
Past Perfect	high + েছিলাম	high + েছিলে	high + েছিলি	high + েছিলো	high + েছিলেন
Past Habitual	high + তাম	high + তে	high + তিস	high + ত	high + তেন
Future Simple	high + বো	high + বে	high + বি	high + বে	high + বেন

Table 2.3: Class 1 Verb Conjugation (template with high and low stems)

Tense	1st Person	2nd Person (Familiar)	2nd Person (Ordinary)	3rd Person (Ordinary)	2nd/3rd Person (Polite)
Present Simple	low + ি	low	low + িস	low + ে	low + েন
Present Continuous	low + ছি	low + ছো	low + ছিস	low + ছে	low + ছেন
Present Perfect	high + েছি	high + েছো	high + েছিস	high + েছে	high + েছেন
Past Simple	low + লাম	low + লে	low + লি	low + লো	low + লেন
Past Continuous	low + ছিলাম	low + ছিলে	low + ছিলি	low + ছিলো	low + ছিলেন
Past Perfect	high + েছিলাম	high + েছিলে	high + েছিলি	high + েছিলো	high + েছিলেন
Past Habitual	low + তাম	low + তে	low + তিস	low + ত	low + তেন
Future Simple	low + বো	low + বে	low + বি	low + বে	low + বেন

Table 2.4: Class 2 Verb Conjugation

Tense	1st Person	2nd Person (Familiar)	2nd Person (Ordinary)	3rd Person (Ordinary)	2nd/3rd Person (Polite)
Present Simple	high + ই	low + ও	high + স	low + য়	low + ন
Present Continuous	high + চ্ছি	high + চ্ছো	high + চ্ছিস	high + চ্ছে	high + চ্ছেন
Present Perfect	high + য়েছি	high + য়েছ	high + য়েছিস	high + য়েছে	high + য়েছেন
Past Simple	high + লাম	high + লে	high + লি	high + লো	high + লেন
Past Continuous	high + চ্ছি-লাম	high + চ্ছিলে	high + চ্ছিলি	high + চ্ছিলো	high + চ্ছিলেন
Past Perfect	high + য়েছিলাম	high + য়েছিলে	high + য়েছিলি	high + য়েছিলো	high + য়েছিলেন
Past Habitual	high + তাম	high + তে	high + তিস	high + ত	high + তেন
Future Simple	high + বো	high + বে	high + বি	high + বে	high + বেন

Table 2.5: Class 3 Verb Conjugation

Tense	1st Person	2nd Person (Familiar)	2nd Person (Ordinary)	3rd Person (Ordinary)	2nd/3rd Person (Polite)
Present Simple	low + ই	low + ও	low + স	low + য়	low + ন
Present Continuous	low + চ্ছি	low + চ্ছো	low + চ্ছিস	low + চ্ছে	low + চ্ছেন
Present Perfect	high + য়েছি	high + য়েছো	high + য়েছিস	high + য়েছে	high + য়েছেন
Past Simple	high + লাম	high + লে	high + লি	high + লো	high + লেন
Past Continuous	low + চ্ছিলাম	low + চ্ছিলে	low + চ্ছিলি	low + চ্ছিলো	low + চ্ছিলেন
Past Perfect	high + য়েছিলাম	high + য়েছিলে	high + য়েছিলি	high + য়েছিলো	high + য়েছিলেন
Past Habitual	low + তাম	low + তে	low + তিস	low + ত	low + তেন
Future Simple	low + বো	low + বে	low + বি	low + বে	low + বেন

Table 2.6: Class 4/5/6 Verb Conjugation

2.4 Adjective

In Bangla, adjectives are words that modify nouns by describing their qualities, states, or attributes. Adjectives in Bangla do not inflect for number, gender, or case, which makes them relatively simple morphologically. For example, the adjective "চালাক (smart)" stays the same in both "চালাক ছেলে " (smart boy) and "চালাক মেয়ে " (smart girl). Adjectives in Bangla typically precede the noun they modify. There are two main types of adjectives in Bangla: qualitative and quantitative. There is no copula "to be (verb)" in Bangla. Some adjectives can form a sentece where the adjective directly follows the noun. For example in "মেয়েটা বুদ্ধিমান। (The girl is intelligent.)", the adjective "বুদ্ধিমান " follws the noun to form a sentence without any verb. This feature distinguishes Bangla from other languages that require a linking verb in such constructions.

Comparison in some languages like English involves comparative and superlative forms of adjectives. Example: good → better → best, small → smaller → smallest. In Bangla, the adjective generally does not change at all. The superlative form adds a word "সবচেয়ে" before the adjective specifying it's in superlative form. A few adjectives from Sanskrit language create the comparative and superlative form by adding "তর" and "তম" respectively after the adjective.

Adjective defined in GF syntax,

```
LinA : Type = SS ;
LinA2 : Type = LinA ;

mkAdj : Str -> LinA = \str -> {s = str} ;

AdjPhrase : Type = LinA ;
```

Since Adjectives in Bangla are invariant, the only necessary information is the string. The type SS refers to a record with only `s string` in GF. The `mkAdj` creates an adjective by taking a `string` input.

`smart_Adj` = `mkAdj` "বুদ্ধিমান";

This line defines the adjective "smart" in bangla. It takes the string form of the adjective in `mkAdj` function.

2.5 Adverb

In Bangla, adverbs are words that modify verbs, adjectives, or other adverbs, adding detail about the manner, time, place, frequency, or degree of an action or quality. For example, in the sentence সে ধীরে হাঁটে ("He/She walks slowly"), the word ধীরে ("slowly") is an adverb modifying the verb হাঁটে ("walks"). Bangla adverbs can be derived from adjectives, nouns, or be standalone words, and they do not inflect for number, gender, or person.

In traditional Bangla grammar books, adverbs are often treated as a subgroup of adjectives rather than as a separate grammatical category. Since Bangla does not always show

clear morphological distinctions between adjectives and adverbs, many have historically grouped them together. For example, the word ভালো ("good/well") can function both as an adjective (ভালো ছেলে – "good boy") and as an adverb (সে ভালো গায় – "he/she sings well").

In GF syntax, Adverbs are defined as type SS that is a record with a `string`. Adverb is defined the same way as Adjective.

2.6 Postposition

In Bangla, postpositions are grammatical elements that function similarly to prepositions in English, but they appear after the noun, pronoun or noun phrase they modify. Unlike English, where one says "on the table," Bangla expresses this as "টেবিলের উপরে". Postpositions in Bangla often follow a genitive or objective noun form, requiring the noun to take an appropriate case marker.

Bangla postpositions do not form a closed word class. With few exceptions, many are derived from nouns in the locative case or perfective participle verb forms, and the boundary between postpositions and other grammatical forms is often fluid. Despite this, it is both practical and useful to treat postpositions as a distinct word class. Many of the locative nouns or perfective participle verbs have changed or expanded their meaning in their use as postpositions. A small number of Bangla postpositions are underived and original.

Common Bangla postpositions include উপরে (above/on), নিচে (under), আগে (before), হতে (from), পর্যন্ত (until), জন্যে (for), মত (like), ধরে (during), and হয়ে (through). Among these, উপরে, নিচে, and আগে are derived from nouns, while হতে, ধরে, and হয়ে originate from verbs. In contrast, পর্যন্ত, জন্যে, and মত are examples of underived or original postpositions.

Postposition defined in GF syntax,

```
oper
  LinPrep : Type = {
    s : Str ;
    c : Case ;
  } ;

  mkPrep = overload {
    mkPrep : Str -> Prep = \s -> lin Prep {s = s ; c = Gen} ;
    mkPrep : Str -> Case -> Prep = \s, c -> lin Prep {s = s ; c = c} ;
  } ;
```

LinPrep defines the linearization type of a prep (postposition) as a record with:

- `s` — the `string` form of the postposition.
- `c` — the grammatical `case` the postposition assigns to the noun.

`mkPrep` is a constructor function for postposition. It is overloaded. The first form creates a postposition from a `string` and assumes the default `case` to be genitive (`Gen`), which reflects Bangla structure where most postpositions require the noun in genitive case. The second form allows explicitly specifying the required `case`.

```
above_prep = mkPrep "উপরে";

before_prep = mkPrep "আগে" Nom;
```

The first defination creates a postposition with the default genitive case. The second defination explicitly sets the required case to nominative, overriding the default.

2.7 Conjunction

Conjunctions are joining words that connect sentences or parts of sentences. In Bangla, conjunctions are classified into three main types: coordinating, subordinating, and correlative conjunctions.

- Coordinating Conjunction - Coordinating conjunctions join elements of equal grammatical rank, such as words or independent clauses. এবং (and), অথবা (or), কিন্তু (but), জন্য (because), অতএব (therefore), তারপর (then) are examples of coordinating conjunctions in bangla language.

  আমি তোমার বাসায় যাবো অথবা আমার বাসায় দেখা করতে পারব।
  I will go to your house or we can meet at mine.

  Here, a sentence pair demonstrates a coordinating conjunction joining two sentences together both in English and Bangla.

- Subordinating Conjunctions - Subordinating conjunctions in Bangla are used to link dependent clauses to main clauses, introducing conditions, reasons, time, and purpose. যে (that), যাতে (so that), যেন (as if) are examples of subordinating conjunctions in bangla language.

  আমি একটা চাকরি খুঁজছি যাতে আমি এই বাড়ি ছেড়ে যেতে পারি।
  I am looking for a job so that I can leave this house.

  An example sentence pair using subordinating conjunction.

- Correlative Conjunctions - Correlative conjunctions in Bangla appear in paired forms. These conjunctions work together to relate two parts of a sentence, emphasizing balance or conditional relationships. Examples include হয় - না হয় (either - or), যদি - তাহলে (if - then), যদিও - তবুও (even though), and না - না (neither - nor).Not all pairs of correlative conjunctions in Bangla have an equivalent pair in English.

  আমি টাকাও পাব না, স্বীকৃতিও পাব না।
  I will neither get the money nor the recognition.

  যদিও আমি তাকে সাহায্য করেছি, তবুও সে আমাকে সাহায্য করেনি।
  Even though I helped him, he did not help me.

Two example sentence pairs using correlative conjunctions. The first pair has an an equivalent pair in English but the second pair does not.

Conjunction in GF syntax stayed the same as generated by the template. Conjunction is a record of a string in Bangla RGL. It was not necessary for generating wikidata description.

2.8 Number System

2.8.1 Digit

Bangla digits, are a set of ten symbols used to represent numbers in the Bangla script. They follow the decimal system. Bangla numbers have two forms - cardinal and ordinal. The numbers from 1-10 have irregular ordinal form. The ordinal form for the rest of the number just has a suffix "তম" added at the end.

Cardinal	০ (0), ১ (1), ২ (2), ৩ (3), ৪ (4), ৫ (5), ৬ (6), ৭ (7), ৮ (8), ৯ (9)
Ordinal	০তম (0th), ১ম (1st), ২য় (2nd) ৩য় (3rd), ৪র্থ (4th), ৫ম (5th), ৬ষ্ঠ (6th), ৭ম (7th), ৮ম (8th), ৯ম (9th)

Table 2.7: Bangla Digits

```
param
  Number = Sg | Pl ;
  CardOrd = NCard | NOrd ;

oper
  LinDig : Type = {s : CardOrd => Str ; n : Number} ;

lincat
  Dig = LinDig ;   -- single digit  -

lin
  -- : Dig -> Digits ;        --   (For single digit number)
  IDig d = d ;
  -- : Dig -> Digits -> Digits ; --    (For multiple digits number)
  IIDig d e = {
  s = table {
    NCard => d.s ! NCard ++ BIND ++ e.s ! NCard ;
    NOrd => d.s ! NCard ++ BIND ++ e.s ! NCard ++ BIND

    ++ "তম" ;

  } ;
  n = Pl ;
```

```
} ;

oper

  mkDig : Str -> Str -> Number -> LinDig = \s1, s2, n -> {
    s = table {
      NCard => s1 ;
      NOrd  => s2
      } ;
    n = n ;
    } ;

lin

  PosDecimal d = { s = d.s ! NCard ; hasDot = False ; n = Pl } ;

  NegDecimal d = { s = "-" ++ BIND ++ d.s ! NCard ; hasDot = False ;
                   n = Pl } ;

  IFrac d i = {
      s = d.s ++ if_then_Str d.hasDot BIND (BIND++"."++BIND)
        ++ i.s ! NCard ;
      hasDot=True;
      n = Pl
    } ;
```

The following GF code defines Bangla digits and their behavior in both cardinal and ordinal contexts. Each digit is represented as a `LinDig` record, which contains a table mapping between `CardOrd` (cardinal or ordinal forms) and a `Str`, as well as a grammatical number (`Number`). Each digit is declared using variable (`D_0` to `D_9`) with its corresponding Bangla character and its ordinal and cardinal form using `mkDig` function. Note that the grammatical number is marked as `Sg` (singular) for digits ০ and ১, and `Pl` (plural) for the rest.

The system also includes linearization rules for multi-digit numbers leveraging the `IIDig` constructor. The `IIDig` constructor is used to concatenate two digits to form a larger number. For example, combining `D_8` and `D_7` would produce the string representation for the number ৮৭ (87)in bangla. The constructor also handles ordinal formatting by appending the Bangla ordinal suffix তম. Associated helpers like `PosDecimal`, `NegDecimal`, and `IFrac` make decimal formatting (positive, negative, and fractional) possible for the system.

2.8.2 Numeral

This section defines the structure and linearization rules for expressing numerals in Bengali within the Grammatical Framework (GF). Unlike the `Digit` module, which models

numeric forms such as 1, 2, 3, the `Numeral` module focuses on their written counterparts- এক, দুই, তিন—as well as their ordinal forms- প্রথম, দ্বিতীয়, তৃতীয়.

This code is designed for handling Bangla numerals, with a focus on both cardinal and ordinal number expressions in a linguistically accurate and computationally structured way. A large part of this code was implemented for Hindi in this paper [8].

```
param
  DForm = unit | ten ;
  DSize = sg | r2 | r3 | r4 | r5 | r6 | r7 | r8 | r9 ;
  Size = singl | less100 | more100 ;

oper
  LinDigit = {s : DForm => Str ; size : DSize ; n : Number ; ord : Str} ;

lincat
  Digit = LinDigit ;
  Sub10 = {s : DForm => Str ; size : DSize ; n : Number ; ord : Str} ;
  Sub100 = {s : Str ; size : Size ; n : Number ; ord : Str} ;
  Sub1000 = {s : Str ; s2 : Str ; size : Size ; n : Number ; ord : Str} ;
  Sub1000000 = {s : Str ; n : Number ; ord : Str} ;

lin num x0 = {
  s = table {
    NCard => x0.s ;
    NOrd  => x0.ord
  } ;
  n = x0.n
} ;


oper mkDigits : Str -> Str -> Str -> DSize -> LinDigit =
  \two, twenty, second, sz ->
  {s = table {unit => two ; ten => twenty } ;
   size = sz ; n = Pl ; ord = second } ;

oper
  mkR : (a1,_,_,_,_,_,_,_,a9 : Str) -> DSize => Str =
   \a1,a2,a3,a4,a5,a6,a7,a8,a9 -> table {
     sg => a1 ;
     r2 => a2 ;
     r3 => a3 ;
     r4 => a4 ;
     r5 => a5 ;
     r6 => a6 ;
     r7 => a7 ;
     r8 => a8 ;
```

```
    r9 => a9
  } ;

rows : DSize => DSize => Str ;
```

The code defines several key parameter types to categorize numeral forms. For instance, `s : DForm => Str` is a table that stores unit and ten forms of a digit:

- `unit` — used when the digit is in a unit position (e.g., দুই for ২).
- `ten` — used when the digit is in the tens position (e.g., বিশ for ২০).

`size : DSize` indicates which digit it is in multi-digit numbers (e.g., `r2` for ২, `r3` for ৩). `n : Number` encodes grammatical number, typically `Pl` to agree with plural constructions, with Sg reserved for ১ and ০. `ord : Str` holds the ordinal form of the digit (e.g., "দ্বিতীয়" for ২য়). `Size` captures broader magnitude categories such as singular, less than 100, or more than 100.

Core linearization categories (`lincats` like `Digit`, `Sub10`, `Sub100`, `Sub1000` and `Sub1000000` represent number expressions at increasing orders of magnitude. They include fields for surface form `strings (s)`, grammatical number `(n)`, and their corresponding ordinal forms `(ord)` allowing both regular and irregular numeral formations to be handled uniformly.

The `LinDigit` type abstract over digits with additional details about form and usage context. The function `mkDigits` is a constructor that builds entries for individual digits, mapping each to its cardinal form (e.g., "দুই" for 2), its ten-form (e.g., "বিশ" for 20), and its ordinal form (e.g., "দ্বিতীয়" for second), while tagging them with appropriate `DSize` values.

A particularly irregular range in Bangla lies in the numbers 11–99, where simple concatenation of digits does not produce the correct result. For example, instead of composing "তিন" (3) and "ত্রিশ" (30) to say "ত্রিশ-তিন" (33) like English, Bangla uses "তেত্রিশ. "To address this, the grammar defines a matrix-like lookup table `rows : DSize => DSize => Str` named `rows` that returns the correct forms.

Beyond 99, numerals follow regular morphological patterns. For example, ৭২৩ is formed by attaching শ suffix to the numeral for ৭ and then combining it with the form for ২৩ . Similar rules apply for thousands and lakhs. To model this, a series of compositional functions—e.g., `pot0`, `pot1`, etc.—build larger numbers from smaller subcomponents. Ordinal forms are generated by appending the suffix তম to the cardinal form.

This modular and layered approach allows the grammar to cover both the lexical exceptions and the productive patterns in Bengali numeral construction.

2.9 Symbol

Certain runtime values such as symbols (`symb`), integers, and floats are treated as linguistic entities in Bangla and need to be integrated into the grammar. In the RGL, these

values are mapped to proper nouns (PN) so they can participate in syntactic constructions such as noun phrases. For details on how proper nouns are handled in general, see section 3.2.

```
mkPN_onRuntimeToken : Str -> LinPN = \str -> {
  s = table {
    Gen => str

   ++ "এর" ;

    _ => str
  } ;
  n = Pl ;
} ;
```

This function creates a proper noun (LinPN) by appending the word "এর" to the input string in the genitive case, and leaving the string unchanged in all other cases. It also assigns an inherent plural number, which reflects the typical interpretation of such tokens as representing sets, quantities, or abstract categories. The resulting noun behaves like any other proper noun in downstream syntactic constructions.

3

Syntax

This chapter describes the syntax of Bangla grammar and its implementation in Grammatical Framework. Only the parts necessary for generating Bangla descriptions for Wikidata items was implementated.

3.1 Common Noun (CN)

Common nouns in GF typically represent countable or mass entities, and their linearization type often mirrors that of simple nouns. In the Bangla RGL, the `LinCN` type is defined as an alias of `LinN`, indicating that at this stage of the grammar, common nouns do not require more complex structures like additional modifiers.

```
LinCN : Type = LinN
```

This definition suffices for `CN` constructions, especially when the primary goal is to support noun usage in simple descriptive sentences for resources such as Wikidata.

3.2 Proper Noun (PN)

Proper nouns represent specific named entities. In Bangla, proper nouns often have an inherent number (usually singular). The `LinPN` type captures this by including:

- A mapping from grammatical case (`Case`) to `string`, as proper nouns may appear in different roles (nominative, genitive, etc.).
- An explicit number (`Number`), though typically proper nouns are singular (`Sg`). Some nouns like "United States" are plural, but still considered proper nouns because they name specific, unique entities.

```
oper
    LinPN : Type = {
    s : Case => Str ;
    n : Number ;
  } ;

  mkLinPN : Str -> Number -> Animacy -> LinPN = \s,n,a -> {
    s = \\c => (mkLinN a s).s ! Inflection n NoArticle c ;
```

```
    n = n
  } ;

  mkPN = overload {
    mkPN : Str -> PN = \s -> lin PN (ResBen.mkLinPN s Sg Inanimate) ;
    mkPN : Number -> Animacy -> PN = s, n, a ->lin PN (ResBen.mkLinPN s n a) ;
  } ;
```

Proper nouns can have different suffixes for a case depending on if they are inanimate or animate. The default animacy and number is `Inanimate` and single (`Sg`) as they are more common. Otherwise, animacy and number is used as input alongside string to create a proper noun. This design ensures that proper nouns like "ঢাকা" (Dhaka) are correctly inflected for case (e.g., nominative "ঢাকা", locative "ঢাকাতে") and maintain their number consistency.

3.3 Quantifiers, Determiners, and Numerals

To support noun phrases in Bangla grammar, the RGL defines dedicated linearization types for quantifiers, determiners, and numerals. Each of these types encodes information necessary for agreement, word order, and article handling in Bangla noun phrases.

3.3.1 Quantifiers

Quantifiers are words that express quantity without being tied to specific numerical values. In the Bangla RGL, the linearization type `LinQuant` includes:

- `s` — the `string` representation of the quantifier
- `a` — the `Article` associated with it

```
LinQuant : Type = {
  s : Str ;
  a : Article ;
} ;

mkQuant : Str -> Article -> LinQuant = \s, article -> {
  s = s ;
  a = article
};
```

3.3.2 Determiners

Determiners provide information about definiteness and quantity, including items such as এই ("this"), ওই ("that"), and numerically-inflected determiners like এই তিনটি ("these three"). The `LinDet` type captures:

- `s` — the `string` representation of the determiner

- `n` — the `Number` (singular/plural) it agrees with
- `a` — the `Article` associated with it

```
LinDet : Type = {
  s : Str ;
  n : Number ;
  a : Article ;
} ;

mkDet : Str -> Number -> Article -> LinDet = \str, num, article -> {
  s = str ;
  n = num ;
  a = article ;
} ;
```

3.3.3 Numerals (Num)

Numerals are number words like এক (”one”), দুই (”two”), তিন (”three”) and are vital for expressing exact quantities. The `LinNum` structure stores:

- `s` — the `string` representation of the numeral
- `n` — the `Number` (singular/plural)

```
LinNum : Type = {
  s : Str ;
  n : Number ;
} ;

mkNum : Str -> Number -> LinNum = \str, num -> {
  s = str ;
  n = num ;
} ;
```

There is already a linearization type for Numerals called `LinNumeral` in section 2.8.2 but in GF, Numeral and Num play different roles. ”Numeral” is used for number parsing and generation and ”Num” is grammar-integrated numeral that is used for constructing noun phrases and other grammatical entities. Other types like Digit, Card are also cast into the abstract category ”Num”. There is a `mkNum : Numeral -> Num` function in GF that maps a Numeral to Num.

3.4 Noun Phrase (NP)

In the Bangla RGL, Noun Phrases (NP) can originate from common nouns, proper nouns, or pronouns. Among these, only pronouns inherently carry person information (e.g., 1st, 2nd, 3rd person), while nouns—both common and proper—are generally treated as

3rd person (Ordinary). This assumption simplifies agreement patterns in predicates and determiners.

To reflect this, The `LinNP` structure stores:

- `s` — A mapping from grammatical case (`Case`) to `string`
- `n` — the grammatical `Number` (singular/plural)
- `p` - the grammatical `Person` (usually `P3H1` for nouns)

```
LinNP : Type = {
  s : Case => Str ;
  n : Number ;
  p : Person
} ;
```

The default rendering of an NP (`linNP`) uses the nominative case (`Nom`), as that's the most common.

```
linNP : LinNP -> Str = \np -> np.s ! Nom ;
```

An `emptyNP` is defined as a helper function. It is used in other functions like `DetCN`.

```
emptyNP : LinNP = {
  s = \\_ => [] ;
  n = Sg ;
  p = P3H1 ;
} ;
```

3.5 Constructing Noun Phrases

In the RGL, noun phrases (NP) can be constructed from a variety of sources. This section describes three primary constructors: `DetCN`, `MassNP`, and `ApposNP`.

```
DetCN : Det -> CN -> NP

DetCN det cn = emptyNP ** {
  s = \\c => det.s ++ cn.s ! Inflection det.n det.a c
} ;
```

This function builds a full noun phrase from a determiner and a common noun. It is one of the most common constructions for Bangla noun phrases:

- `det.s` contributes the determiner `string`, such as এই ("this") or ওই ("that").
- `cn.s ! Inflection det.n det.a c` inflects the noun for number and article based on the determiner and applies the appropriate case.
- The resulting NP inherits default 3rd-person features from `emptyNP`.

Examples: এই বইটা (this book), একজন শিক্ষক (a teacher)

```
MassNP : CN -> NP

MassNP cn = emptyNP ** {
  s = \\c => cn.s ! Inflection Sg NoArticle c
} ;
```

This constructor creates an NP from a bare mass noun or any noun used without a determiner. The noun is assumed to have singular number and no article, suitable for mass nouns. Examples: চিনি (sugar), পানি (water)

```
ApposNP : NP -> NP -> NP

ApposNP np1 np2 = np1 ** {
  s = \\c => np2.s ! Nom ++ bindComma ++ np1.s ! c
} ;
```

This constructor handles appositive noun phrases, where one NP explains or renames another.

- `np2.s ! Nom` places the second NP in nominative case.
- `bindComma` inserts a comma(,) between the two parts).
- `np1.s ! c` places the first noun phrase in the requested case.

Examples: রবীন্দ্রনাথ, একজন কবি (Rabindranath, a poet) ; ঢাকা, বাংলাদেশের রাজধানী (Dhaka, the capital of Bangladesh)

3.6 Constructing Common Nouns

In the RGL, CN (common noun phrases) can be built not only from lexical nouns but also via composition with modifiers, possessives, or apposition structures. This section describes the main constructors used to build complex CNs.

```
UseN : N -> CN ;
UseN n = n ;
```

This is the simplest constructor that lifts a bare noun into the `CN` category via an identity funtion.

```
ApposCN : CN -> NP -> CN ;

ApposCN cn np = cn ** {
  s = \n => np.s ! Nom ++ cn.s ! n
} ;
```

The `ApposCN` function combines a common noun with an appositive noun phrase. This construction is useful for cases where a named entity is described by its type:

- The `np` is rendered in nominative case and placed before the common noun.
- The `cn` is placed last and any case applied to the entity only changes the `cn`.

Example: রাজা জন স্নো (King Jon Snow)

```
PossNP : CN -> NP -> CN ;
PossNP cn np = cn ** {
  s = \n => np.s ! Gen ++ cn.s ! n
} ;
```

The PossNP function attaches a possessive noun phrase to a common noun to form a common noun.

- The possessor np is rendered in genitive case (Gen) and placed before the noun.
- The cn is placed last and any case applied to the entity only changes the cn.

Examples: কারো বই (book of someone), আমাজনের সাপ (snake of Amazon)

```
AdjCN : AP -> CN -> CN ;
AdjCN ap cn = {
  s = \n => ap.s ++ cn.s ! n ;
  numSuff = cn.numSuff ;
} ;
```

An adjective phrase be added before a common noun to create a new common noun.

- The adjective string is prepended to the noun.
- The numSuff field is preserved from the original CN for inflection.

Examples: বড় বাড়ি (big house), বাদামি গরু (brown cow)

```
AdvCN : CN -> Adv -> CN ;
AdvCN cn adv = {
  s = \n => cn.s ! n ++ adv.s ;
  numSuff = cn.numSuff ;
} ;
```

An adverbial modifier is applied to a common noun to modify its meaning or specificity.

- The adverbial string is placed after the noun form. In Bangla, this can also be added before the noun. It was placed after because it better suits our application grammar.
- The numSuff field is preserved from the original CN for inflection.

Example: বাড়ি পাহাড়ের উপরে (house on the hill)

3.7 Constructing Adverbial phrases (Adv)

Adverb phrases in Bangla can be built from prepositional phrases or by composing multiple adverbial modifiers. These constructions allow nuanced temporal, spatial, or manner expressions.

```
PrepNP : Prep -> NP -> Adv ;

PrepNP prep np = {
  s = np.s ! prep.c ++ prep.s
} ;
```

The function PrepNP converts a noun phrase into an adverbial phrase using a postposition.

- The noun phrase is inflected for the case required by the postposition (prep.c).
- The postposition string is appended after the noun.

Examples: বইয়ের ওপর (on the book), ঘরের মধ্যে (inside the house)

```
AdAdv : AdA -> Adv -> Adv ;

AdAdv ada adv = {
  s = ada.s ++ adv.s
} ;
```

This constructor adds an ad-adverbial modifier (e.g., intensifier) to another adverb to create an adverb phrase. Examples: খুব দ্রুত (very quickly), অনেক আস্তে (extremely slowly)

3.8 Constructing Adjective phrases (AP)

```
AdvAP : AP -> Adv -> AP ;
AdvAP ap adv = ap ** {
  s = adv.s ++ ap.s ;
}
```

Example: স্বভাবতই নরম (soft by nature)

The function AdvAP allows a post-adverb to modify an adjective phrase.

```
PositA a = a ;

ComparA a np = a ** {
  compar = np.s
  } ;
```

AdjOrd ord = { s = "সবচেয়ে"++ ord.s } ; The function PositA and ComparA in GF creates the positive and comparative form of the adjective. In Bangla, the adjective a remains unchanged. The function AdjOrd creates the superlative form by adding "সবচেয়ে" before the adjective.

3.9 Constructing Utterance-Level Noun Phrases (Utt)

These functions lift noun phrases and common nouns into utterance-level phrases, choosing a surface form appropriate for final output.

```
UttNP : NP -> Utt ;

UttNP np = {
s = np.s ! Nom
} ;
```

`UttNP` extracts the nominative form of an `NP`, suitable for utterances or sentence subjects.

```
UttCN : CN -> Utt ;

UttCN cn = UttNP (MassNP cn) ;
```

`UttCN` first converts a `CN` to a noun phrase using `MassNP`, assuming no article, and then extracts the nominative form.

These are useful for producing simple sentences, queries, or stand-alone entity descriptions where a fully inflected noun phrase is needed.

4

Description Generation

This chapter describes the process of generating Bangla descriptions for Wikidata items using the Bangla RGL.

4.1 Lexicon Collection

In order to generate natural language descriptions for Wikidata entities in Bangla, a lexicon of translated entity names was required. While some items in Wikidata already had Bangla labels, the vast majority lacked them. Manually translating over 100,000 items was infeasible, so a semi-automated approach was adopted.

A Python script was developed to batch-translate the missing labels using the Google Translate API using `googletrans` library. Items were prepared as a newline-separated list of entity names, which were then translated in bulk. This significantly reduced the time and effort compared to translating each item individually.

There was some concern about translation fidelity, particularly whether Google Translate would alter or misinterpret the structure of the input or reorder lines. To mitigate this, the output was manually inspected at key intervals — including the beginning, end, and several samples from the middle. The line alignment was preserved, and the translations were found to be largely accurate.

The resulting translated lexicon now serves as the basis for generating Bangla descriptions for Wikidata entities in the grammar.

4.2 Human Descriptions

The `HumanDescriptionsBen` module is responsible for generating Bangla descriptions for Wikidata items classified as human. It composes various lexical components drawn from `CountriesBen` and `ProfessionsBen` and syntactic components drawn from Bangla RGL to construct fluent noun phrases.

The module defines several key categories relevant to human descriptions:

- `Person`: a record containing the person's name and their gender. The gender is helpful for languages that have grammatical gender. Here, it helps choosing the correct term for professions with gendered terms.

- `Professions`: a function from gender to noun, to support gender-appropriate inflection (e.g., for gendered terms like অভিনেতা (actor) / অভিনেত্রী (actress)).
- `BirthAndDeathYears`: an `Adv` phrase encoding life span (e.g., (১৯০১ – ১৯৮৭)).
- `Bornplace`: a prepositional adverbial phrase marking the place of birth.

Implementation

Lifespan is constructed using optional constructors such as:

```
BornAndDied b d =  ParadigmsBen.mkAdv ("(" ++ b.s ++ "–" ++ d.s ++ ")") ;
OnlyDied d = ParadigmsBen.mkAdv ("( –" ++ d.s ++ ")") ;
OnlyBorn b = ParadigmsBen.mkAdv ("(" ++ b.s ++ "– )") ;
NoBirthOrDeath = ParadigmsBen.mkAdv ("") ;
```

These allow flexible handling of incomplete data such as only knowing the death year. Birthplace is added using a custom preposition:

```
Bornin country = SyntaxBen.mkAdv born_in_Prep country.s ;

born_in_Prep : Prep = mkPrep "জন্মগ্রহণকারী" locative ;
```

This attaches the locative form of the country with the word "জন্মগ্রহণকারী", functioning as a marker for birthplace.

Each person is constructed with:

```
PersonBuilding str gen = {s = str.s ; g = gen.g} ;
```

This ensures that gender information is carried through for agreement in profession.

Descriptions are realized as NPs. For instance:

```
SameNationalityBuilding p c prof birthtime = mkNP (mkCN
(mkCN c.nationality (mkCN (prof ! p.g))) birthtime) ;
```

The `SameNationalityBuilding` function constructs a noun phrase when a person's nationality and birthplace are the same. It builds a hierarchical noun phrase combining nationality, profession, and lifespan information in a natural order.

The structure is as follows:

- The innermost noun phrase is the profession, chosen according to the person's gender (`prof ! p.g`), allowing gender-appropriate inflections (e.g., male or female forms).
- This profession phrase is then combined with the nationality noun (`c.nationality`), forming a compound noun phrase that describes the person's national identity and profession together.
- Finally, the lifespan phrase (`birthtime`) is appended to this compound noun phrase.

This produces a phrase that identifies the person's nationality and profession along with their birth and death years. For example: "বাংলাদেশী চিকিৎসক (১৯৫৫ –)" meaning

“Bangladeshi doctor (1955 –)”. A syntax tree of a sample description is shown in figure 4.1.

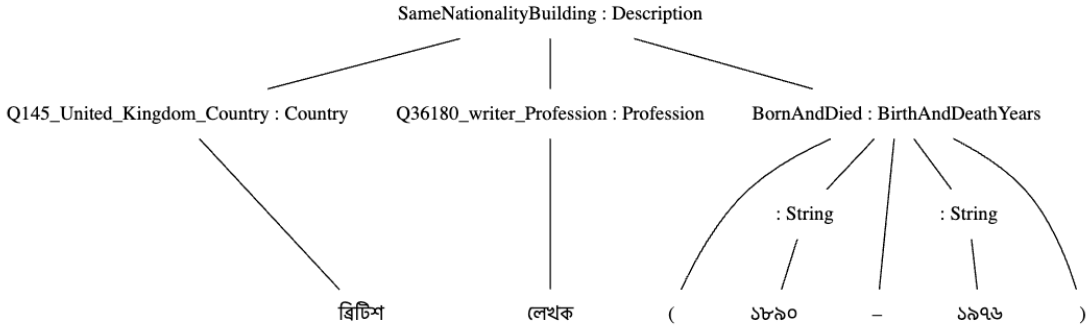

Figure 4.1: Syntax tree of a person’s description with same nationality and birthplace

```
DiffNationalityBuilding p c place prof birthtime = mkNP (mkCN
(mkCN place (mkCN c.nationality (mkCN (prof ! p.g)))) birthtime) ;
```

The `DiffNationalityBuilding` function handles the case where a person’s nationality and birthplace differ. It constructs a noun phrase (`NP`) that explicitly includes both the place of birth and the nationality, followed by the profession and lifespan information. The nested use of `mkCN` (common noun constructors) builds a hierarchical structure reflecting this:

- The innermost phrase is the profession, selected according to the person’s gender (`prof ! p.g`).
- This is combined with the nationality noun (`c.nationality`), forming a compound noun phrase describing the person’s national identity.
- The birthplace noun (`place`) is then combined with this nationality-profession phrase to emphasize that the person was born in a different place than their nationality.
- Finally, the lifespan phrase (`birthtime`) is appended to complete the description.

For example, this can generate a phrase like: “পর্তুগালে জন্মগ্রহণকারী ব্রিটিশ ঐতিহাসিক (১৯৫০ –)” which literally means “British historian born in Portugal (1950 –)”, indicating a person born in Portugal but identified as British. A syntax tree of a sample description is shown in figure 4.2.

When the nationality is unknown or unspecified, the `NationalityUnknown` function omits the nationality component and only includes the profession and lifespan. This simpler phrase still provides meaningful description without making assumptions about nationality:

```
NationalityUnknown p prof birthtime = mkNP (mkCN (mkCN
(prof ! p.g)) birthtime) ;
```

This yields a phrase like: “গণিতবিদ (১৮৯০–১৯৫৫)” - “mathematician (1890–1955)”, where no nationality is specified. A syntax tree of a sample description is shown in figure 4.3.

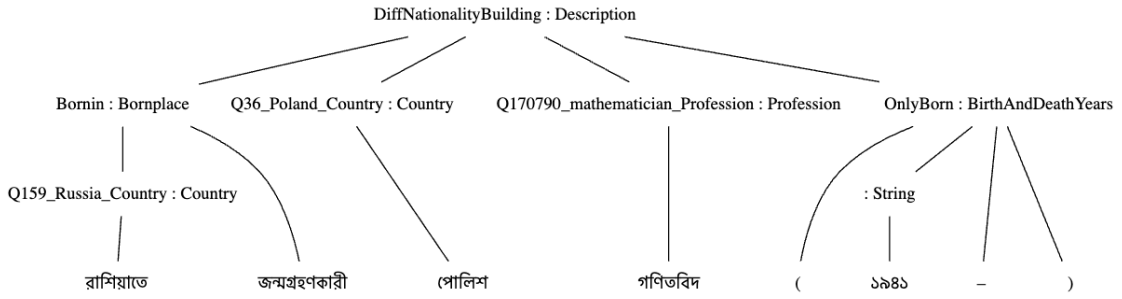

Figure 4.2: Syntax tree of a person's description with different nationality and birthplace

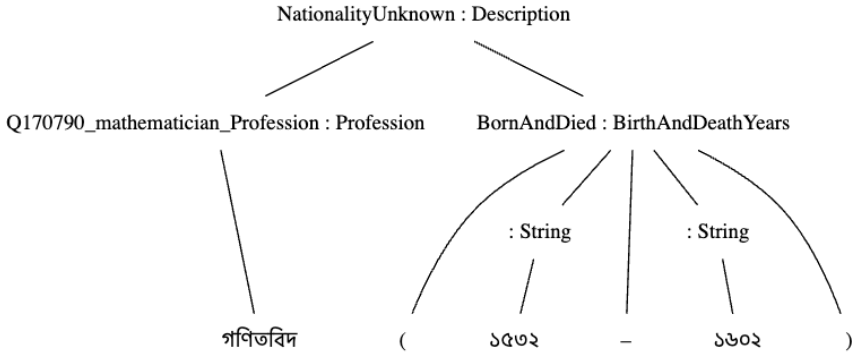

Figure 4.3: Syntax tree of a person's description with unknown nationality

This design ensures flexibility in generating descriptions based on available data, gracefully handling missing or conflicting information.

Together, these components allow for flexible, accurate generation of Bangla descriptions of human entities in Wikidata, capturing attributes like profession, nationality, lifespan, and place of birth. The modular design allows easy extension to handle more attributes (e.g., religion, awards, or aliases) in the future.

4.3 Location Descriptions

The `DescriptionsBen` module generates Bangla descriptions for a variety of geographical entities such as cities, provinces, and countries, as well as institutions like universities and natural features including islands and lakes. This module combines lexical resources from `CountriesBen`, `CitiesBen`, `ProvincesBen`, `CityKindsBen`, and `WaterbodiesBen` with syntactic constructors from the Bangla RGL to produce fluent, grammatically correct phrases.

The module defines the following categories:

- `Location`: A noun phrase (`NP`) representing a geographical place such as a country, city, or province.
- `UniversityKinds`, `IslandKinds`, `WaterKinds`: Common noun categories (`CN`)

used to classify different types of entities.

- `Attribute`: Adverbial modifier (`Adv`) as founding date.
- `Description`: The final output utterance (`Utt`), representing the generated phrase.

Location Construction

Locations can be constructed flexibly using the overloaded function `mkLocation`, which supports either combining two noun phrases or passing a single noun phrase unchanged:

```
mkLocation = overload {
  mkLocation : NP -> NP -> NP = ApposNP ;   -- Combine two locations
  mkLocation : NP -> NP = id NP              -- Single location
} ;
```

While the identity function may seem redundant, it will make the code more robust. This is future proofing the code.

```
CityCountryLocation city country = mkLocation city country.s ;
ProvinceCountryLocation province country = mkLocation province country.s ;
CountryLocation country = mkLocation country.s ;
```

The `mkLocation` function is used to create noun phrases for a variety of locations such as a city in a country, a province in a country or simply a country by itself.

```
  cnUtt : CN -> Utt =
    \cn -> mkUtt (mkNP cn) ;
  locUtt : CN -> NP -> Utt =
    \kind,location -> cnUtt (NounBen.PossNP kind location) ;
```

These operations construct utterance-level (`Utt`) descriptions from common nouns and locations. The `cnUtt` function converts any common noun (`CN`) into a complete utterance (`Utt`) by forming a noun phrase without an article. The `locUtt` function builds location-based descriptions by first creating a possessive construction (using `NounBen.PossNP`) to bind a kind term (e.g., "university") to a location (e.g., "Dhaka"), then passing the result to `cnUtt`. For example, combining বিশ্ববিদ্যালয় (university) with ঢাকা (Dhaka) yields ঢাকার বিশ্ববিদ্যালয় ("Dhaka's university") as a complete utterance. This pattern efficiently generates natural Bangla descriptions where entities are linked to their geographical context.

City Descriptions

Capital City Descriptions

The functions `ProvinceForCapital` and `CountryForCapital` generate descriptive phrases for capital cities by combining the Bangla word for "capital" — "রাজধানী"— with the appropriate location context. These functions build noun phrases that express possession or belonging.

```
ProvinceForCapital province country = locUtt (mkCN
```

```
(mkN "রাজধানী")) (mkLocation province country.s) ;

CountryForCapital country = locUtt (mkCN

(mkN "রাজধানী")) country.s ;
```

The components of the code work as follows:

- The common noun `kind` is created by `mkCN (mkN "রাজধানী")`, which constructs the common noun "capital" for Bangla.
- The location is built differently in the two functions:
- `ProvinceForCapital` combines the province and country noun phrases using `mkLocation`. This creates a nested location like "Guayas Province, Ecuador" - "গুয়াস প্রদেশ, ইকুয়েডর". (combining province name and country name into one location phrase)
- `CountryForCapital` uses the country noun phrase directly as the location.

A syntax tree of such a city description is shown in figure 4.4.

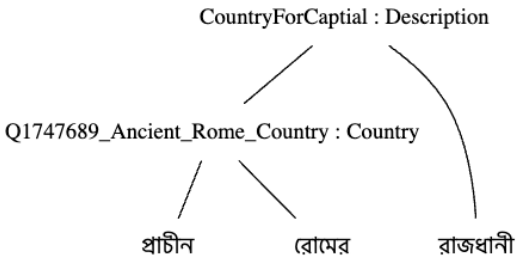

Figure 4.4: Syntax tree of a capital city description

Example output descriptions - "জাপানের রাজধানী" (capital of Japan)
"কম্বোডিয়া, কেপ প্রদেশের রাজধানী" (capital of Kep Province, Cambodia)

Other City Descriptions

```
CityDescription kind location = locUtt kind location ;
```

The other city descriptions are generated the same way. We simply replace `mkCN (mkN "রাজধানী"))` with `kind`. `kind` is already a common noun (`CN`) describing city types such as জেলা (district), পৌরসভা (municipality) . A syntax tree of such a city description is shown in figure 4.5.

University Descriptions

University descriptions in Bangla are generated by combining three elements:

- `kind (CN)` - A common noun that specifies the type of university — general, public, or private.

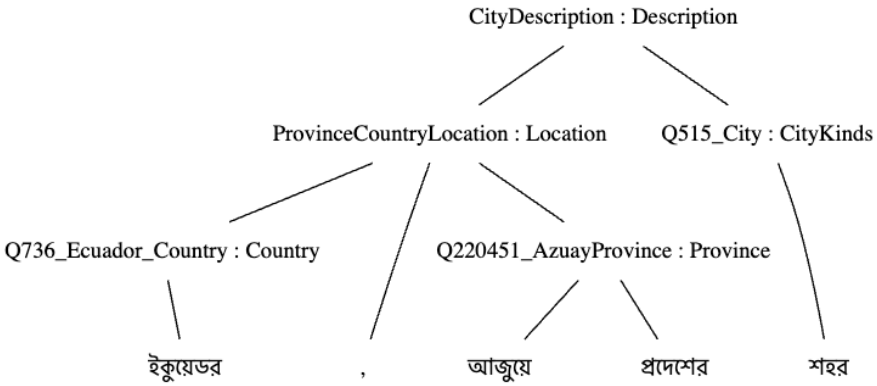

Figure 4.5: Syntax tree of a city description

- `location (NP)` - A noun phrase that describes where the university is located. This could be a standalone country, city, province or a nested phrase.
- `attr (Adv)` - An adverbial modifier used for adding the year of the university was founded.

```
founded_in_year_Prep = mkPrep
  "সালে প্রতিষ্ঠিত" nominative ;

FoundedIn year = mkAdv founded_in_year_Prep (symb year) ;
noAttr = ParadigmsBen.mkAdv "";
```

Founding dates are expressed using a custom adposition phrase `founded_in_year_Prep`. An adverbial phrase is constructed using `mkAdv (PrepNP)` combining `founded_in_year_Prep` and `year NP`. To allow for cases where the year is unknown `noAttr` is provided. This builds an adverbial phrase such as: "১৯২১ সালে প্রতিষ্ঠিত" ("established in 1921")

The core construction function `UniversityDescription` produces a complete noun phrase that expresses a university's type, its affiliation with a location and it's founding year.

```
UniversityDescription kind location attr =
cnUtt (mkCN (NounBen.PossNP kind location) attr) ;
```

The function uses the helper `NounBen.PossNP` to generate a possessive construction such as ঢাকার বিশ্ববিদ্যালয় (Dhaka's University). This is then combined with the founding year into a `CN` using `mkCN` function, and finally into a full utterance using `cnUtt`. There is no problem with a `NP` being empty string so `location` being missing will not cause a problem. If the founding year is missing, `noAttr` can be used instead. So, the description generation of university is flexible. A syntax tree of a university description is shown in figure 4.6.

Example output: ভিয়েতনামের বিশ্ববিদ্যালয় (university in Vietnam), মার্কিন যুক্তরাষ্ট্রের প্রাইভেট বিশ্ববিদ্যালয় ১৮৫৭ সালে প্রতিষ্ঠিত (private university in United States founded in 1857), দক্ষিণ কোরিয়া, সিওলের বিশ্ববিদ্যালয় (university in Seoul, South Korea)

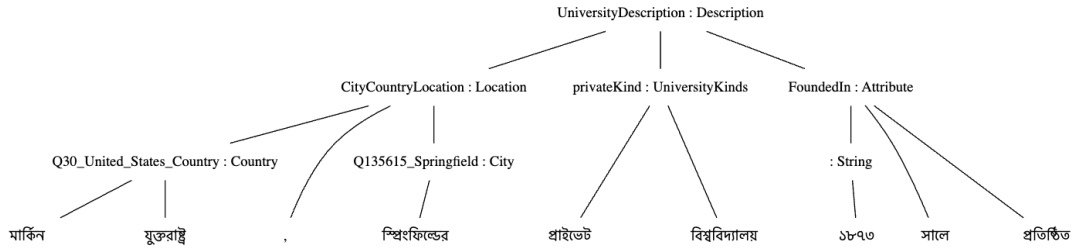

Figure 4.6: Syntax tree of a university description

Island Descriptions

The `DescriptionsBen` module includes constructions to describe islands, optionally specifying the water body they are located in.

```
Island = mkCN (mkN
      "দ্বীপ") ;

IslandDescription water kind location = cnUtt (mkCN (NounBen.PossNP
      kind location) (mkAdv located_prep water)) ;
NoWaterIslandDescription kind location = locUtt kind location ;

  located_prep = mkPrep
    "অবস্থিত" locative ;
```

The common noun (`CN`) `Island` is used for kind when no other `kind` is found. This ensures that even if `water, kind. location` are all empty strings, there will still be a description. All other functions are already explained in the sections above. The difference between `IslandDescription` and `NoWaterIslandDescription` lies on whether the surrounding water body (a sea, a river...) is included as part of the location context. Syntax trees for both of them are shown in figure 4.7 and figure 4.8.

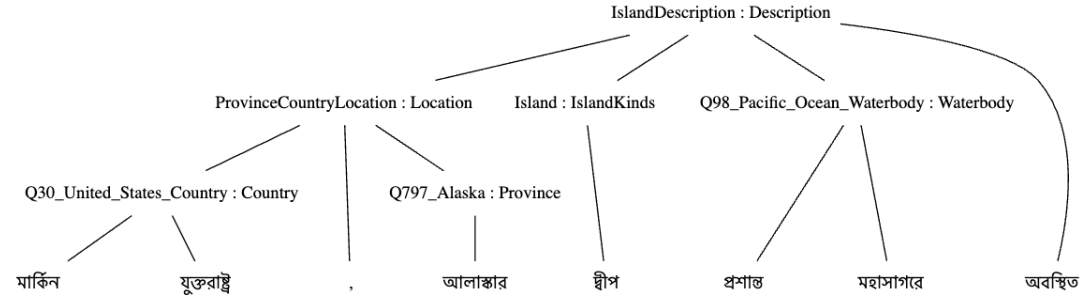

Figure 4.7: Syntax tree of an island description

Example output: পাপুয়া নিউগিনির দ্বীপ পাপুয়ার উপসাগরে অবস্থিত (island in Papua New Guinea, located in Gulf of Papua), সুইডেন, নরবটেন কাউন্টির দ্বীপ (island in Norrbotten County, Sweden)

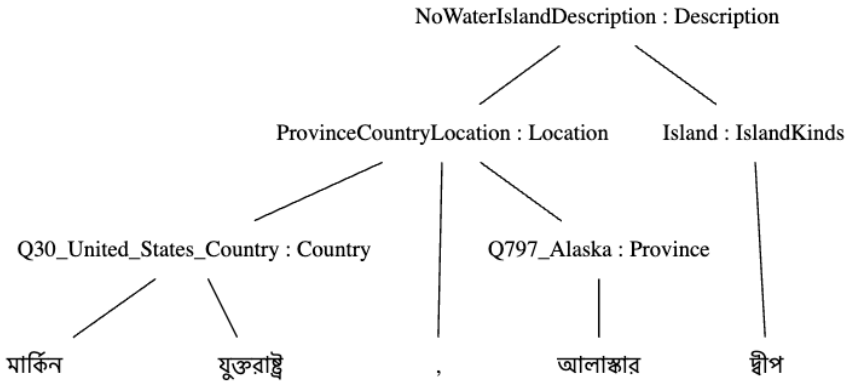

Figure 4.8: Syntax tree of an island description without surrounding water body information

Lake Descriptions

Lakes are described using a simple construction that binds a lake name to its geographic location. The core noun is defined as:

```
Lake = mkCN (mkN
      দীঘি ) ;

  LakeDescription lake location = locUtt lake location ;
```

This creates the common noun দীঘি (lake), used in the description function as `lake`. The `locUtt` operation will generate possessive phrases. Syntax trees of lake description is shown in figure 4.9.

Example output: সুইজারল্যান্ড, বার্নের দীঘি (lake in Bern, Switzerland)

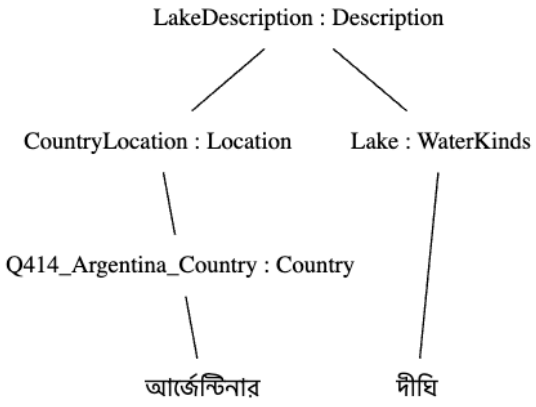

Figure 4.9: Syntax tree of a lake description

This module provides comprehensive coverage of geographical and institutional descriptions in Bangla. It supports both simple cases and complex hierarchical location structures, ensuring natural word order and grammatical correctness. The modular design

facilitates easy extension with additional entity types by adding new lexical entries and construction patterns in the application grammar.

5

Results

This chapter presents the evaluation of the generated Wikidata entity descriptions for Bangla. The evaluation focuses on multiple criteria, including grammatical correctness, lexical accuracy, consistency with the input data, coverage ratio and adhering with Wikidata's multilingual content generation goals.

The generated descriptions were produced using a prototype Bangla resource grammar built GF. To assess the quality of these outputs, we performed both manual and automatic evaluation.

Additionally, we provide a qualitative analysis highlighting common sources of error, and areas where the grammar performs well or requires further refinement. These results aim to inform future improvements in the Bangla RGL and demonstrate the feasibility of controlled multilingual text generation for low-resource languages.

Where applicable, the performance of the generated Wikidata entity descriptions is contextualized by comparing it to existing entity descriptions in Wikidata.

5.1 Evaluation Criteria

To comprehensively assess the quality of the generated Bangla descriptions for Wikidata entities, we evaluate the outputs based on five key criteria: grammatical correctness, lexical accuracy, coverage ratio, factual accuracy, and consistency to descriptions of other languages. These criteria are both linguistic and task-specific, ensuring that the generated texts are not only well-formed in Bangla but also faithful to the semantic intent of the structured data they are derived from.

- Grammatical Correctness - This criterion examines whether the generated sentences conform to the rules of Bangla grammar. Evaluations under this criterion consider aspects such as subject-verb agreement, word order, morphological inflection and syntactic structure.
- Lexical Accuracy - Lexical accuracy measures whether the correct Bangla words are used in descriptions to express the intended. Errors in this category include incorrect lexical choices, mistranslations, or use of overly generic or ambiguous vocabulary.
- Factual Accuracy - This criterion evaluates the semantic faithfulness of the generated descriptions to their source data in Wikidata. It assesses whether all key

facts (subject, predicate, object relations) are correctly and completely reflected in the output without omissions or hallucinations.

- Coverage Ratio - The coverage ratio measures the proportion of Wikidata entities for which the application grammar is capable of generating well-formed natural language descriptions for Bangla. This criterion is crucial for understanding the practical applicability of the grammar and identifying limitations in its syntactic or lexical scope.
- Multilingual Consistency - Multilingual consistency evaluates the extent to which the generated Bangla descriptions align with generated entity descriptions in other languages. Wikidata aims to provide uniform, high-quality content across a diverse range of languages so maintaining semantic and stylistic consistency is essential for ensuring that information is equally accessible and accurate across linguistic boundaries.

Together, these criteria provide a holistic view of the system's performance.

5.2 Evaluation Methods

To assess the quality and reliability of the generated Bangla descriptions for Wikidata entities, we adopted a multi-faceted evaluation approach. Each evaluation criteria — grammatical correctness, lexical accuracy, factual accuracy, coverage ratio, and multilingual consistency — needs to be assessed using methods suited to its nature and feasibility within a large-scale generation task. The evaluation strategy for each criterion are given below.

The use of the Grammatical Framework (GF) ensures that all generated descriptions follow the syntactic rules defined in the Bangla resource grammar. Since GF guarantees that every output conforms to the grammar's abstract syntax and corresponding linearization rules, we assume all generated sentences to be grammatically well-formed.

Lexical entries were mostly populated using automatic translation via Google Translate, with English Wikidata labels as input. In cases where the original label was not in English or translation was unavailable, the term retained in its original form. Given the scale of the lexicon (over 100,000 entries), a full manual validation is impractical. Instead, a manual sampling is conducted to evaluate translation quality, identifying general patterns and common issues. Fully automatic evaluation was deemed infeasible due to lack of ground-truth translations.

The generation system constructs descriptions directly from Wikidata API. As such, the generated outputs are assumed to be factually accurate as long as the input data in Wikidata is correct. Since the grammar only linearizes already structured semantic data, it does not introduce new factual content or reinterpret information. Any factual inaccuracies in output are therefore attributed to errors or inconsistencies in the original Wikidata entries rather than the generation system.

To measure how extensively the Bangla grammar can generate valid descriptions across entity types, we compute the coverage ratio by batch-generating descriptions for all

available Wikidata categories. We then compare:

- The number of entities with existing Bangla descriptions.
- The number of entities for which generation succeeded.
- Cases where a Bangla description existed originally but generation failed.
- Cases where generation succeeded but no original Bangla description was available.

This analysis highlights both the utility and limitations of the current grammar implementation.

We evaluate multilingual consistency by comparing the generated Bangla descriptions with corresponding machine-generated descriptions in other languages — specifically English and Chinese. Sentence embeddings are obtained using the `paraphrase-multilingual-mpnet-base-v2` model [10] from HuggingFace. Cosine similarity score is computed between embeddings of Bangla-English and Bangla-Chinese sentence pairs. Additionally, we calculate the same similarity scores for the original Wikidata descriptions in these languages, allowing us to benchmark the closeness of generated descriptions to existing multilingual content. This comparison informs us how semantically consistent the Bangla output is in relation to other languages supported by Wikidata.

5.3 Evaluation Results

Since grammatical correctness and factual accuracy is assumed to be true due to the way GF works, in this section we will evaluate lexical accuracy, coverage ratio and multilingual consistency of the generated Bangla descriptions.

Due to the large number of lexicons, exhaustive verification was not possible. However, only a very small number of translation failures were observed. These cases occurred because the original entities were expected to be in English, as per the Wikidata entry structure, but some were in other languages or contained non-English characters. The translation script assumes all input to be in English. This was done deliberately to avoid inconsistencies from automatic language detection, which can often misidentify proper nouns or mixed-language entries. As a result, entity names not originally in English (e.g., Oebisfelde-Weferlingen, Mühlheim an der Donau) remained untranslated, preserving their original form. Some entries were only partially translated, such as সাংস্কৃতিক heritage তিহ্য D-2-7242-0083 ওয়ালারডর্ফে, derived from “Cultural heritage D-2-7242-0083 in Wallersdorf”, and হার্মোপলিস পারভা (7th ম নোম)from “Hermopolis Parva (7th nome)”. Additionally, certain characters not recognized as English, such as the letter å in ”Grenå”, were left unchanged in the output: গ্রেন å. In addition to the issues mentioned, there were occasional typical machine translation errors, such as awkward phrasing or incorrect word choices, but these were relatively infrequent and did not significantly impact overall usability. Despite these anomalies, the majority of translations were accurate and usable.

The generated descriptions were compared against the existing original Bangla entries across all categories: cities, universities, islands, lakes, and humans. For islands and lakes and humans, 10,000 entries were randomly sampled due to time constraints and larger dataset sizes.

Entity Type	Total Entries	Original Entries	Generated Entries	Only Generated	Only Original
City	32,496	6,625 20.39%	32,440 99.83%	25,824 79.47%	9 0.03%
University	14,184	955 6.73%	14,155 99.80%	13,202 93.08%	2 0.01%
Island	10,000	2,984 29.84%	9,997 99.97%	7,013 70.13%	0 0.00%
Lake	10,000	8608 86.08%	10,000 100.00%	1,392 13.92%	0 0.00%
Human	10,000	3,771 37.71%	9,985 99.85%	6,217 62.17%	3 0.03%
Total	76,680	22,943 29.92%	76,577 99.87%	53,648 69.96%	14 0.02%

Table 5.1: Coverage Comparison of Original and Generated Bangla Descriptions

Using a multilingual sentence embedding model, embedding vector is created for sentence pairs and their consine similarity is calculated. Sentence pairs with higher consine similarity reside near the same embedding space and are more likely to mean the same thing. A mean is taken of all scores for each sentence pairs.

Entity Type	Original (EN-BN)	Generated (EN-BN)	Original (ZH-BN)	Generated (ZH-BN)
City	0.3222	0.6604	0.5394	0.7923
University	0.6987	0.8472	0.7386	0.8149
Island	0.6758	0.6597	0.7136	0.7142
Lake	0.3186	0.4660	0.6252	0.7128
Human	0.6989	0.6809	0.7111	0.8023

Table 5.2: Cosine Similarity Scores Between Bangla Descriptions and Their English/Chinese Counterparts

5.4 Discussion

The evaluation results demonstrate that the Bangla descriptions generated using the GF-based resource grammar exhibit strong coverage and consistency across various Wiki-

data entity types. Nearly complete coverage was achieved for all categories, with especially high generation rates for entries that previously lacked Bangla descriptions—such as universities and cities. This underscores the system's capability to substantially enrich multilingual content for low-resource languages like Bangla. The description generation failed in a few cases because all necessary information was not available. The fields chosen for description generation are fields with high frequency but it is still possible for them to be missing. Island and Lake categories reached full coverage because there is a default description for them in the absence of information.

To properly evaluate the similarity score, a baseline need to established. For the sentence-pair with the same meaning in English and Bangla, the model gave a similarity score of 0.8059. For a sentence-pair with slightly different meaning, the model gave a similarity score of 0.6848. For unrelated sentence-pair, the model gave a similarity score of 0.2415. From this, we can see that the scoring for the original is much worse than expected. The sentence pairs for the original English and Bangla descriptions are unrelated for city, lake and human categories. The rest of the original descriptions scores indicate partial matching. The high cosine similarity scores for generated descriptions, indicate that the generated outputs maintain semantic alignment with their source descriptions across languages. This further validate the generalizability of this approach.

Nevertheless, a few limitations were observed. A small number of translation failures arose due to the assumption that input labels were in English. While this assumption avoided the complexity and potential inconsistency of automatic language detection, it led to untranslated or partially translated entries when encountering non-English entity names. Some machine translation artifacts—such as awkward phrasing or untranslated characters—also persisted but were generally rare and did not significantly affect the utility of the output.

5.5 Future Work

While the current system provides a robust foundation for generating Bangla descriptions of Wikidata entities, several limitations remain that offer promising directions for future work.

First, the Bangla Resource Grammar Library (RGL) used in this project lacks many syntactic constructs necessary for expressing a wider range of sentence types. In particular, the current grammar does not include interrogative constructions, nor does it support imperative verb forms in the present or future tense. A lot of syntax rules are also not implemented yet. Adding support for these would greatly increase the expressive power of the system and enable the generation of more diverse and complex sentence structures.

Secondly, lexicon collection can also be further refined. Better handling of multilingual input labels—particularly entity names in non-English scripts—will also improve translation completeness. Integrating a more robust language detection module could address this.

Finally, expanding the system to support a broader range of entity types beyond the current categories (cities, universities, islands, lakes, and humans) would further demon-

strate the generalizability of the approach. Incorporating descriptions for entities such as organizations, historical events, artworks, or scientific concepts would test the flexibility of the grammar and lexicon, and help move toward comprehensive multilingual coverage of Wikidata.

6

Conclusion

This work presents a GF-based Bangla resource grammar capable of generating structured and semantically accurate descriptions for various Wikidata entities. The system achieves high coverage across diverse domains and demonstrates superior consistency when compared to existing Bangla entries. Evaluation through cosine similarity against English and Chinese sources indicates strong multilingual alignment, confirming the effectiveness of grammar-based generation for low-resource languages.

Despite a few translation challenges related to multilingual input labels, the overall quality and coverage of generated content affirm the utility of GF as a robust tool for multilingual knowledge base enrichment. The proposed grammar thus offers a scalable pathway for addressing the content gap in Bangla Wikidata entries and opens the door for broader linguistic expansion in similar low-resource languages.

Bibliography

[1] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase", *Commun. ACM*, vol. 57, no. 10, pp. 78–85, Sep. 2014, issn: 0001-0782. doi: 10.1145/2629489. [Online]. Available: https://doi.org/10.1145/2629489.

[2] A. Ranta, "Grammatical framework", *Journal of Functional Programming*, vol. 14, no. 2, pp. 145–189, 2004.

[3] A. Ranta, *Grammatical framework: Programming with multilingual grammars*. CSLI Publications, Center for the Study of Language and Information Stanford, 2011, vol. 173.

[4] A. Ranta, "The GF Resource Grammar Library", *Linguistics in Language Technology*, vol. 2, 2009. [Online]. Available: http://elanguage.net/journals/index.php/lilt/article/viewFile/214/158.

[5] A. Ranta, "Embedded controlled languages", in *Controlled Natural Language: 4th International Workshop, CNL 2014, Galway, Ireland, August 20-22, 2014. Proceedings 4*, Springer, 2014, pp. 1–7.

[6] A. Ranta, "Multilingual text generation for abstract wikipedia in grammatical framework: Prospects and challenges", *Logic and Algorithms in Computational Linguistics 2021 (LACompLing2021)*, pp. 125–149, 2023.

[7] K. Angelov, A. Carrión del Fresno, E. Voloshina, and A. Ranta, "Leveraging grammatical framework and wordnet for natural language generation from wikidata", in *International Symposium on Distributed Computing and Artificial Intelligence*, Springer, 2024, pp. 173–184.

[8] S. Virk, *Computational linguistics resources for Indo-Iranian languages*. 2014.

[9] H.-R. Thompson, *Bengali: A comprehensive grammar*. Routledge, 2020.

[10] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks", in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Nov. 2019. [Online]. Available: http://arxiv.org/abs/1908.10084.

A

Appendix 1

Script for translating lexicon terms,

```
from googletrans import Translator
import time
import sys

CHAR_LIMIT = 4000  # Google Translate character limit per request

def read_lines(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        return f.readlines()

def write_lines(file_path, lines, mode='a'):
    with open(file_path, mode, encoding='utf-8') as f:
        f.writelines(lines)


def process_translation(input_file, output_file, src='auto', dest='en'):
    lines = read_lines(input_file)
    total_lines = len(lines)
    translator = Translator()

    current_idx = last_written_line = 930

    while current_idx < total_lines:
        batch = []
        char_count = 0

        # Build batch without exceeding char limit
        while current_idx < total_lines:
            line = lines[current_idx]
            if char_count + len(line) > CHAR_LIMIT:
                break
            batch.append(line)
            char_count += len(line)
            current_idx += 1
```

```
        try:
            string = ("").join(batch)
            translated_lines = translator.translate(string, src=src, \
            dest=dest).text
            translated_lines += '\n'
            write_lines(output_file, translated_lines)
            last_written_line = current_idx
        except Exception as e:
            print(f"Error during translation at line {last_written_line}: {e}")
            break

        time.sleep(1)  # Rate limiting

    print(f"Translation completed up to line {last_written_line}.")


if __name__ == "__main__":
    if len(sys.argv) != 3:
        print(f"Usage: python {sys.argv[0]} <input_file> <output_file>")
        sys.exit(1)

    input_file = sys.argv[1]
    output_file = sys.argv[2]

    lines = read_lines(input_file)
    process_translation(input_file, output_file, src='en', dest='bn')
```

Script for calculating cosine similarity,

```
import json
import sys
from sentence_transformers import SentenceTransformer, util
import numpy as np

model = SentenceTransformer('paraphrase-multilingual-mpnet-base-v2')

def calculate_similarity(sentence_pair):
    embeddings = model.encode(sentence_pair, convert_to_tensor=True)
    return util.pytorch_cos_sim(embeddings[0], embeddings[1])[0][0].item()

def calculate_cosine_similarity(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        data = [json.loads(line) for line in file]
    i = 0
    orig_pair_list = []
    gen_pair_list = []
```

```
    orig_pair_list_2 = []
    gen_pair_list_2 = []
    total = len(data)
    for i in range(total):
        item = data[i]
        if item['orig_bn'] and item['gen_bn'] \
        and item['orig_en'] and item['gen_en']:
            orig_pair_list.append([item['orig_bn'], item['orig_en']])
            gen_pair_list.append([item['gen_bn'], item['gen_en']])
        if item['orig_bn'] and item['gen_bn'] \
        and item['orig_zh'] and item['gen_zh']:
            orig_pair_list_2.append([item['orig_bn'], item['orig_zh']])
            gen_pair_list_2.append([item['gen_bn'], item['gen_zh']])
        i = i + 1
    orig_score = []
    gen_score = []
    for index in range(len(orig_pair_list)):
        orig_score.append(calculate_similarity(orig_pair_list[index]))
        gen_score.append(calculate_similarity(gen_pair_list[index]))

    print(f"Mean of the cosine score of the original descriptior pair \
    (English and Bangla): {np.mean(orig_score)}")
    print(f"Mean of the cosine score of the generated descriptior pair \
    (English and Bangla): {np.mean(gen_score)}")

    orig_score = []
    gen_score = []
    for index in range(len(orig_pair_list_2)):
        orig_score.append(calculate_similarity(orig_pair_list_2[index]))
        gen_score.append(calculate_similarity(gen_pair_list_2[index]))
    print(f"Mean of the cosine score of the original descriptior pair \
    (Chinese and Bangla): {np.mean(orig_score)}")
    print(f"Mean of the cosine score of the generated descriptior pair \
    (Chinese and Bangla): {np.mean(gen_score)}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print(f"Usage: python {sys.argv[0]} <input.jsonl>")
        sys.exit(1)
    calculate_cosine_similarity(sys.argv[1])
```

Script for calculating coverage,

```
import json
import sys

def calculate_coverage(file_path):
```

```
    with open(file_path, 'r', encoding='utf-8') as file:
        data = [json.loads(line) for line in file]
    total = len(data)
    original_cover = total - sum(1 for d in data if \
    not d["orig_bn"] or d["orig_bn"].strip() == "")
    gen_cover = total - sum(1 for d in data if \
    not d["gen_bn"] or d["gen_bn"].strip() == "")
    gen_yes_original_no = sum(1 for d in data if \
    (not d["orig_bn"] or d["orig_bn"].strip() == "") and d["gen_bn"])
    gen_no_original_yes = sum(1 for d in data if \
    (not d["gen_bn"] or d["gen_bn"].strip() == "") and d["orig_bn"])
    print(f"Total entries: {total}")
    print(f"Original Bangla descriptions \
    {original_cover} - {original_cover/total:.2%}")
    print(f"Gen Bangla descriptions \
    {gen_cover} - {gen_cover/total:.2%}")
    print(f"Gen covers that original do not \
    {gen_yes_original_no} - {gen_yes_original_no/total:.2%}")
    print(f"original covers that gen do not \
    {gen_no_original_yes} - {gen_no_original_yes/total:.2%}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print(f"Usage: python {sys.argv[0]} <input.jsonl>")
        sys.exit(1)
    calculate_coverage(sys.argv[1])
```