Reversing Hurford's Packing Strategy using Arithmetic Criteria - A Numeral Decomposer for Incremental Unsupervised Grammar Induction

Isidor Konrad Maier *maier@b-tu.de*
Chair of Communication Engineering
Brandenburg University of Technology Cottbus – Senftenberg
DE-03046, Cottbus/Chóśebuz, Siemens-Halske-Ring 14, Deutschland

Matthias Wolff *matthias.wolff@b-tu.de*
Chair of Communication Engineering
Brandenburg University of Technology Cottbus – Senftenberg
DE-03046, Cottbus/Chóśebuz,Siemens-Halske-Ring 14, Deutschland

Abstract

This paper presents a novel numeral decomposer based on arithmetic criteria. It enables a new automatic learning process for numeral grammars that is universally applicable to all languages, as it is based on fundamental, language-independent arithmetic properties. Specifically, the arithmetic criteria depend on Hurford's Packing Strategy but not on a base-10 assumption. Hurford's Packing Strategy constitutes numerals by *packing* factors and summands to multipliers. We found out that a numeral of value n has a multiplier larger than $\sqrt{n}$, a summand smaller than $n/2$ and a factor smaller than $\sqrt{n}$. Using these findings, the numeral decomposer attempts to detect and *unpack* factors and summands in order to reverse Hurford's Packing Strategy. We tested applicability for incremental unsupervised grammar induction in 257 languages. In this way, we obtained grammars with sensible mathematical attributes that explain the structure of numerals. The grammars induced by the numeral decomposer are often close to expert-made and more compact than numeral grammars induced by the modern state-of-the-art grammar induction tool GITTA. Furthermore, this paper contains a report about the few cases of incorrectly induced mathematical attributes, which are often linked to linguistic peculiarities like context sensitivity.

Keywords: Numeral Words, Hurford's Packing Strategy, Numeral Decomposition, Incremental Grammar Induction, Context Sensitivity in Numerals.

1. INTRODUCTION

1.1 Motivation and Related Work

Text normalization tasks like detecting and forming complex numerals correctly consistently pose a challenge to neural networks (Sproat, 2022). Therefore, numeral grammars are often programmed manually (Akinadé & Ọdẹ́jọbí, 2014; Khamdamov et al., 2020; Rhoda, 2017). Numeral grammar induction is the way to automate the programming. The following numeral grammar induction approaches exist.

- Hammarström (2008) proposed a method that subdivides a set of numerals into k-sized clusters based on a similarity measure. Then, generalizations can be made inside the clusters.
- Flach et al. (2000) made a proposal for automatic learning of finite-state numeral grammars.

- Beim Graben et al. (2019) proposed that numerals may be added to a lexicon until a boundary is reached. Then, a penalty signal arises that urges the learner increasingly to summarize several numerals in a generalization.

In a broader sense, the topic of numeral grammar induction is related to the linguistic theory of numerals and their computational modeling on the one hand, and to general grammar induction of natural language on the other hand.

Regarding the linguistic theory of numerals, Brainerd (1966) collected seven studies on numeral grammars in different languages and attempted to draw general conclusions. Our work is mainly based on Hurford (2007). His theory is outlined in more detail in his book (Hurford, 2011). Other works on numeral morphology include Zabbal (2005), Veselinova (2020), and Žoha et al. (2022). Ionin and Matushansky (2006) discusses the morphology of complex numerals in the context of morphosyntax, i.e., the relation of the morphology to the sentence structure. Other related sources on numeral morphosyntax include Ivani (2017) and Martí (2020). Derzhanski and Veneva (2018) give a summary of exceptional phenomena in the structures of numerals. Specifically, for the number 58, Derzhanski (2025) describes structures of the numeral in 720 diverse languages. The study has been conducted based on WALS (Dryer & Haspelmath, 2013), a large database about structural properties of languages including chapters about numerals (Gil, 2013a, 2013b; Stolz & Veselinova, 2013). Andersen (2004) discusses implications of the structure of numerals in various languages on the question whether or not all humans use an universal grammar. Mendia (2018) and Anderson (2019) discussed epistemic numbers, i.e., generalized number phrases like 'twentysome'.

Grammar induction is a wide field of research that refers to the process of learning formal grammars from data. It arose in the 1990s (Carroll & Charniak, 1992; Klein & Manning, 2001; Stolcke & Omohundro, 1994). A systematic and detailed review of the literature on unsupervised grammar induction till 2019 was performed by Muralidaran et al., 2021. Notably, only 1 out of 33 reviewed studies presented an incremental grammar learning method, namely Seginer (2007). Since 2018, significant advancements have been made, particularly with deep learning and neural models, leading to more effective and scalable grammar induction techniques. In particular, Kim et al. (2019) showed that a neural parametrization of marginal dependencies enhances the induction of probabilistic context-free grammars. Shen et al. (2019) tested LSTMs with ordered neurons on a variety of tasks related to grammar induction. Other significant works on neural approaches include Htut et al. (2018) and Drozdov et al. (2019). Three of the newest tools for natural language grammar induction are GITTA (Winters & Raedt, 2020), ShortcutGrammar (Friedman et al., 2022), and LanguageLearner (Jon-And & Michaud, 2024). In this work, we use GITTA as a baseline method among Hammarström (2008), and Derzhanski and Veneva (2020). GITTA induces context-free grammars by using the Wagner-Fischer algorithm (Wagner & Fischer, 1974) to create common templates for similar expressions. Lately, Li et al. (2024) and Zhao et al. (2025) argued that heterogeneous data including vision or speech in addition to text can improve grammar induction.

Our experiments show that our symbolic arithmetic-based approach outperforms state-of-the-art grammar induction approaches in numeral grammar induction, since it utilizes special sophisticated knowledge about the structure of numerals.

1.2 Overview

This work employs both, inductive and deductive, reasoning. Inductively, we establish a theory of the arithmetical relations between subnumerals and an idea how the theory can be applied in an algorithm to decompose numeral words. Deductively, we test and enhance the algorithm for the task of grammar induction in 257 natural languages. The developed numeral decomposer is supposed to reverse Hurford's Packing Strategy (Hurford, 2011). The Packing Strategy—which is explained in Section 2 in more detail—constitutes a numeral word by packing 0, 1 or 2 numerals to a base morpheme M. In English, examples for base morphemes M are 'teen', 'ty', 'hundred', and 'thousand'. The numerals packed to M must be interpreted either by addition—in which case we call them summands—or by multiplication—in which case we call them factors. Example: For the English numeral 'two hundred sixty', two numerals are packed to the base morpheme 'hundred', 'two' as a factor, and 'sixty' as a summand. Therefore, our numeral decomposition algorithm is supposed to unpack the subnumerals 'two' and 'sixty' when parsing the input $(260,$'two hundred sixty'), and therefore, the desired output would be '_ hundred _'$(2, 60)$. In this regard, the algorithm works similarly to a part-of-speech-tagger, stemmer or parser (compare Alkhazi (2019), Sumamo and Teferra (2018), and Chorozoglou et al. (2021), respectively).

In Section 3 we specify the objectives of the numeral decomposer.

The algorithm requires knowledge about the parsed numeral's number value, as well as a lexicon of number-numeral pairs that allows to recognize subnumerals. The algorithm evaluates found subnumerals based on arithmetic criteria presented in Section 4. Based on the criteria, it decides whether or not to unpack them. When assuming that the numeral must follow a base-10 system, criteria for decomposing are well known. One can calculate the decimal digits of the number value n as $\lfloor n/10^{k-1} \rfloor (mod\ 10)$ for $k = 1,2, \ldots$ and detect the numeral words of the digits inside n's numeral word (compare Graben et al. (2019)).

However, we do not assume a certain base system. Instead, we mainly rely on our finding that factors and summands of a numeral N cannot have more than half of N's value. Therefore, when N's value is n, being $\leq n/2$ is a necessary criterion for a subnumeral of N to be unpacked. Only this necessary criterion is used for a basic numeral decomposer that we present in Section 5.1. It works in standard cases, but it can fail if the numeral uses an unusual order of subnumerals, or if a certain critical subnumeral—such as 'veinte' in 'veintiuno'—is not contained letter-by-letter. These details are described in Section 5.2. Extra unpacking criteria are established for an advanced numeral decomposer algorithm that fixes most errors of the basic version. The advanced algorithm is presented in Section 5.3.

In Section 6, we discuss the performance of both numeral decomposer versions by reviewing induced grammars in 257 languages.

1.3 Notations and Wordings

In this subsection we establish our notation for numbers and numeral words.

Specific numbers are normally written with Hindu-Arab digits. For number variables, we use lower case letters. If X is a numeral word, then $n(X)$ denotes the number of X. Often, we will also denote $n(X)$ by X's lower-case letter x.

Specific numerals or strings are written in quotation marks. The empty string is denoted by ε. For numeral or string variables, we use capital letters. If x is any kind of number expression, then $N(x)$ denotes the numeral of x in the language dealt with[1]. Often, we will also denote $N(x)$ by x's upper-case letter X. By $\mathcal{N}$, we denote the set of numeral words of natural numbers $\mathbb{N}$ in the language dealt with[2].

Examples:	Numbers	Numerals
specific	6	'six'
variable	$x = 100$	X ='one hundred'
dependent	$n($'six'$\cdot X) = 600$	$N(6 + x)$ ='one hundred and six'

[1] For the sake of simplicity, we assume that there is one unambiguous spelling for each numeral. Deviating spellings or names may be considered part of another language (variety).

[2] Numeral words only exist for a finite set of natural numbers in most languages, so $\mathcal{N}$ and $\mathbb{N}$ do not have a one-to-one correspondence.

By '·' we denote the concatenation of strings. We use the same words for number relations to also describe relations between the respective numerals. The wordings

"Numeral X
is larger than / is smaller than / equals / is a divisor of / is a multiple of
numeral Y"

mean that the numbers x and y have the respective relation. This allows us to describe arithmetic relations between two numerals or between a numeral and a number with less effort.

Note that "Numeral X is larger than numeral Y" should not be interpreted as if Y is a substring of X. In order to describe string relations between numerals we will only use the wordings 'is a substring/subnumeral/superstring/supernumeral of' or 'contains'/'is contained in'.

As mentioned before, the decomposer unpacks certain subnumerals out of a numeral. Suppose that in a numeral X the subnumerals $U_1, \ldots, U_k$ are unpacked. This implies that $X = S_1 \cdot U_1 \cdot S_2 \cdot \ldots \cdot U_k \cdot S_{k+1}$ with strings S_i. Then we present the decomposition as

$$X = S_1\_S_2\_\ldots\_S_{k+1}(U_1, \ldots, U_k)$$

where the _ denote placeholders. The term $S_1\_S_2\_\ldots\_S_{k+1}$ can be seen as an epistemic number expression that would be spoken $S_1 \cdot$ some $\cdot S_2 \cdot$ some... $\cdot S_{k+1}$ (compare Mendia, 2018 and Anderson, 2019). In the following, $S_1\_\ldots\_S_{k+1}$ is interpreted as a function of numeral words, defined on a domain $\mathcal{D} \subset \mathcal{N}^k$:

$$S_1\_\ldots\_S_{k+1}: \mathcal{D} \to \mathcal{N}, (U_1, \ldots, U_k) \mapsto S_1 \cdot U_1 \cdot S_2 \cdot U_2 \cdot S_3 \cdot \ldots \cdot U_k \cdot S_{k+1} \quad (1)$$

We call $S_1\_\ldots\_S_{k+1}$ the template or template function of the decomposition. Alternatively, when x and $u_1, \ldots, u_k$ are the numbers of the numerals X and $U_1, \ldots, U_k$, we can present the decomposition with the numbers as

$$x = S_1\_S_2\_\ldots\_S_{k+1}(u_1, \ldots, u_k)$$

The notation implies that $S_1\_\ldots\_S_{k+1}$ can be interpreted as a number function on $\mathbb{D} \subset \mathbb{N}^k$:

$$S_1\_\ldots\_S_{k+1} : \mathbb{D} \to \mathbb{N}, (u_1, \ldots, u_k) \mapsto n(S_1 \cdot N(u_1) \cdot S_2 \cdot \ldots \cdot N(u_k) \cdot S_{k+1}) \quad (2)$$

Example: In the English (en_GB)[3] numeral X = 'twenty-seven thousand and two hundred and six', the subnumerals $N(27)$ and $N(206)$ can be unpacked. Then, we present the decomposition as

X = _ thousand and _('twenty-seven','two hundred and six'), or

27206 = _ thousand and _$(27, 206)$.

The resulting numeral function is

_ thousand and_ : $\{N(d) \mid d = 1,\ldots,999\}^2 \to \mathcal{N}, (U_1, U_2) \mapsto U_1 \cdot$ thousand and $\cdot\ U_2$,

[3] In parentheses, we mention the names of our datasets of a newly mentioned language, if they deviate from the mentioned name.

and the number function is

$$\_\text{ thousand and }\_ : \{1,\ldots,999\}^2 \to \mathbb{N}, (u_1, u_2) \mapsto n(N(u_1)\ \cdot \text{thousand and} \cdot\ N(u_2)).$$

English speakers know that $n(N(u_1) \cdot \text{thousand and} \cdot N(u_2))$ means $1000 * u_1 + u_2$. For the general case, however, such an arithmetical equation is not trivial to find.

2. AXIOMS BASED ON HURFORD'S PACKING STRATEGY

First, we briefly summarize the explanation of the Packing Strategy from Hurford (2007). Hurford says: ”The Packing Strategy is a universal constraint on numeral systems. It applies very widely to developed numeral systems. It is not a truism, but exceptions are rare. The Packing Strategy operates in conjunction with a small set of phrase structure rules, which are shared by all developed numeral systems.” These rules are given in Fig. 1.

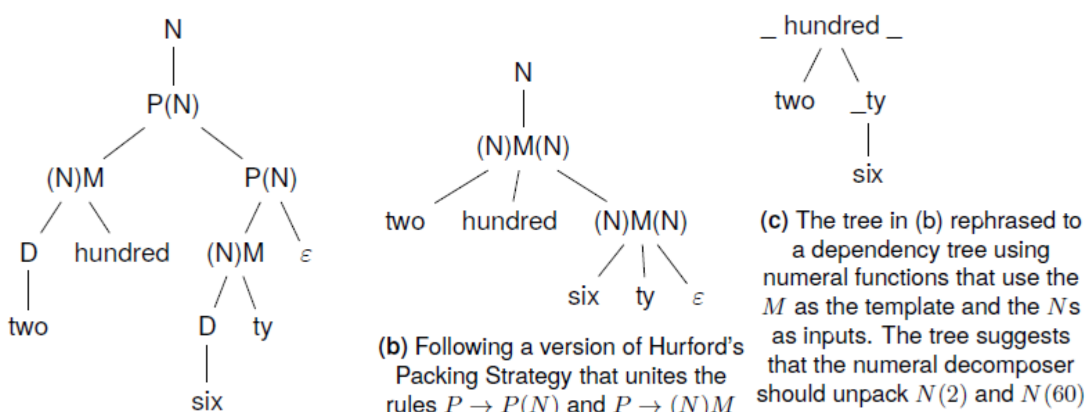
Hurford also mentioned that

- in each rule, ”the sister constituent of NUMBER must have the highest possible value”.
- ”the Packing Strategy says nothing about linear order, but only about the hierarchical dominance relationships between constituents of numeral expression”.

$$\text{Number} \to \begin{Bmatrix} \text{Digit} \\ \text{Phrase (NUMBER)} \end{Bmatrix} \quad \text{(Interpreted by addition)}$$

$$\text{Phrase} \to \text{(NUMBER) M} \quad \text{(Interpreted by multiplication)}$$

FIGURE 1: Graphic originally from Hurford, 2007. Curly brackets indicate ’either/or’ options, Parentheses indicate optional choices. DIGIT is the category of basic lexical numerals, such as in English ’one’, ..., ’nine’. M is the category of multiplicative base morphemes, such as in English ’ty-’, ’teen’, ’hundred’, ’thousand’, or ’million’.



(a) Packing the English numeral ’two hundred sixty’.

(b) Following a version of Hurford’s Packing Strategy that unites the rules $P \to P(N)$ and $P \to (N)M$ to $N \to (N)M(N)$.

(c) The tree in (b) rephrased to a dependency tree using numeral functions that use the M as the template and the Ns as inputs. The tree suggests that the numeral decomposer should unpack $N(2)$ and $N(60)$ when parsing $N(260)$, and then when parsing $N(60)$ it should unpack $N(6)$.

FIGURE 2: Reinterpreting Hurford’s idea of the composition of ’two-hundred sixty’ as a dependency tree.

We establish a new interpretation of the Packing Strategy based on the following axioms. The reinterpretation is also in line with the theories of Zabbal (2005) and Ionin and Matushansky (2006). An example of the English numeral $N(260)$ in Fig. 2 shows the generation of the numeral according to Hurford, as well as according to our reinterpretation.

Axiom 1. *The wording of each compound numeral X implies a calculation of its number value x as $x = fa * mu + su$, in which mu is the value of a multiplicative base morpheme from M (see Figure 1) and fa and su are the "factor" and "summand" numbers.*

Justification. *According to Hurford's Packing Strategy, X is composed of* ***PHRASE (NUMBER),*** *which should be interpreted as $phrase(+number)$. Further,* ***PHRASE*** *is composed of* ***(NUMBER) M****, which means $(number *)m$.*

Combined, we have $X =$ ***(NUMBER) M (NUMBER),*** *which means $(number *)m(+number)$. After renaming the words, we have $X = (FA)MU(SU)$, which means $(fa *)mu(+su)$. If both, FA and SU, exist in X, then our assumption is established.*

If FA is left out, then X means $mu + su$. In this case we interpret it so that X contains FA as an empty string that implies the neutral factor 1. This invisible string does not have an effect on the meaning, since $mu + su = 1 * mu + su$, so it does not hide information either.

Likewise, if SU is left out, then X means $fa * mu$. In this case we interpret it so that X contains SU as an empty string that implies the neutral summand 0. This empty string does not have an effect on the meaning, since $fa * mu = fa * mu + 0$. □

As showcased in the justification of Axiom 1, the supposed calculation $fa * mu + su$ does not mean that the corresponding numeral words FA, MU and SU are contained in X literally. If an implied subnumeral FA, MU or SU is not contained in X literally, we call it a **masked subnumeral**. Subnumerals can be masked for several reasons, including the following.

1. 'one hundred': As already mentioned, if $fa = 1$ *or* $su = 0$, then the implied subnumerals FA and SU can be left out because of redundancy, e.g., in English one simply says 'one hundred' instead of 'one hundred and zero'. In many other languages, the 'one' is also left out.

2. 'thirteen': Implied subnumerals can be subject to grammatical flexion, fusion with adjacent morphemes, or any other phenomenon that causes them to deviate from their standard form. E.g., in the English numeral 'thirteen', both implied subnumerals 'three' and 'ten' are not literal subnumerals because of that.

As a complement to Axiom 1, we assume:

Axiom 2. Each subnumeral of X not containing MU is FA, SU, or a subnumeral of those.

Moreover, we establish a basic assumption on the arithmetical relations between fa, mu and su.

Axiom 3. In the implied calculation $x = fa * mu + su$ of a numeral X, we postulate that $fa < mu$ and $su < mu$.

Justification. *If $fa \geq mu$, then $x \geq u^2$. Therefore, we suspect that a bigger multiplicative base morpheme number mu could have been used, which contradicts that "the sister constituent of NUMBER [FA] must have the highest possible value". If $su \geq mu$, we question why the numeral X would not be formed as $FA' \cdot MU \cdot SU'$ with $n(FA') = fa + 1$ and $n(SU') = su - mu$.* □

The established axioms are used in Section 4 to prove unpacking criteria that can distinguish the subnumerals FA and SU—which are supposed to be unpacked—from other subnumerals.

3. OBJECTIVES

In this section we describe the practical objectives of the numeral decomposer algorithm. In Section 6 we describe to what degree these objectives have been achieved.

We had initially stated that the goal is to design a numeral decomposer that mimics Hurford's Packing Strategy, i.e., the factor word FA and the summand word SU should be unpacked. For the sole purpose of grammar induction, we state our objectives in a more pragmatic way. A flat list of number-numeral pairs

$$\{(1, \text{'one'}), (2, \text{'two'}), \dots (4, \text{'four'}), \dots (14, \text{'fourteen'}) \dots\},$$

is given as the input numeral data set. The numeral decomposer maps each numeral to a template. While for basic numerals like 'one', 'two', or 'four' the template will be the numeral itself, a complex numeral like 'fourteen' is mapped to '_teen'. In either case, the numeral can be reconstructed from the template. In the cases of 'one, 'two', and 'four' the reconstruction is trivial, while in the case of '_teen' the template has to be combined with another template 'four' in order to reconstruct 'fourteen'. Since the templates collectively reconstruct the numerals, the set of templates constitutes a lexicon that—with given grammar rules—generates the input data set.

The lexicon of templates is smaller than the original list, because several numerals can share one template. E.g., if the numbers $14, 16, 17, 19$ are decomposed as

$$14 = \text{\_teen}(4), 16 = \text{\_teen}(6), 17 = \text{\_teen}(7) \text{ and } 19 = \text{\_teen}(9),$$

then four entries of the original list are replaced by one entry in the lexicon of templates. As described in Section 1.3, the template '_teen' can then be described as a function that operates in the domain $\{4, 6, 7, 9\}$. And a template '_ hundred and _' may comprise $9 * 99$ numerals in one entry. Overall, a reduction of lexicon is achieved if numerals are decomposed in a uniform way.

Objective 1 (Compactness). *The numeral decomposer should produce as few different templates as possible.*

However, templates should not be too uniform. Numerals that share a common template should have a reasonable relation. The English numerals 'twenty-one' and 'twenty-seven thousand' could both be mapped to the same template, which may be 'twenty-_' or '_-_', even if their mathematical relation seems unreasonable. We define a reasonable relation in the following way:

Objective 2 (Correctness). *The functional equation of each template function must be affine linear.*

Finally, we comment on the frequent phenomenon of masked subnumerals.

Comment on treatment of masked subnumerals: Masked subnumerals cannot be found or unpacked unless tolerant pattern recognition is involved. We decided not to include tolerance because it would leed to inaccurate grammars. If 'thir' would get unpacked in 'thirty' as if it was $N(3)$, then the decomposition $\text{\_ty}(3) = 30$ would imply that $N(3) \cdot$'ty' = 'threety' is $N(30)$ which is inaccurate. Generally, masked subnumerals usually constitute exceptions, and exceptions are unsuitable for generalization.

4. UNPACKING CRITERIA

Based on our new interpretation of Hurford's Packing Strategy, we establish unpacking criteria, i.e., criteria that distinguish FA, SU and their subnumerals from MU and its supernumerals. The unpacking criteria are mathematically proven under the following working assumption.

Working assumption: *If a numeral N contains another numeral N', then $n' \leq n$.*

We comment on the validity of this assumption at the end of this chapter.

Unpacking Criterion 1 (Necessary Criterion). Let X be a numeral. Let S be a subnumeral of X that is contained in X's factor word FA or in X's summand word SU. Then $2 * s < x$.

Proof. S being contained in FA or SU implies $s \leq fa$ or $s \leq su$. If $s \leq fa$, then

$$2 * s \leq 2 * fa \leq mu * fa \leq fa * mu + su = x$$

Thus, we have $2 * s < x$ unless the numeral system is base 2 and $mu = 2$ and $su = 0$. If $s \leq su$, then

$$2 * s \leq 2 * su = su + su < mu + su \leq fa * mu + su = x.$$

Thus, in either case, $2 * s < x$. □

This criterion alone is sufficient for a basic numeral decomposer, see I. Maier and Wolff, 2022.

Unpacking Criterion 2 (Sufficient Criterion). *Let X be a numeral. Let S be a subnumeral of X that satisfies $s^2 \leq n$. Then, S is contained in X's factor word FA or in X's summand word SU.*

Proof. If S is neither contained in FA nor SU, then by Axiom 2, S contains MU. Therefore we assume that $s \geq mu$. Then, since $fa < mu$ and $su < mu$, we have

$$s^2 \geq mu^2 = mu * mu = (mu - 1) * mu + mu > (mu - 1) * mu + (mu - 1) \geq fa * mu + su = x$$

Hence, S must be contained in either FA or SU. □

The criteria can be used to decide whether or not to unpack subnumerals. The criteria do not only apply to FA and SU directly but also to all sub-subnumerals of those. Thus, in order to use the criteria, one may look for the longest subnumerals that still match the criteria.

The criteria presented so far leave a gap for subnumerals valued between $\sqrt{\mathrm{x}}$ and $\frac{\mathrm{x}}{2}$, for which further unpacking criteria are needed.

Next, we add a classification for subnumerals that are supposed to be unpacked despite not satisfying Unpacking Criterion 2:

Unpacking Criterion 3 (Auxiliary Criterion). *Let X be a numeral. Let S be a subnumeral of X that does not contain X's multiplier word MU, but let $s^2 > x$. Then, S is SU or contained in SU.*

Proof. By Axiom 2, S must be equal or contained in either FA or SU. If it is contained in FA, then $s \leq fa$, hence

$$s^2 \leq fa^2 < fa * mu \leq fa * mu + su = x$$

Thus, S must be contained in SU. □

The criteria 1-3 can be summarized as:

$s < \sqrt{n} \Rightarrow S$ is (contained in) FA or SU
$\sqrt{n} < s < n/2 \Rightarrow S$ can be (contained in) SU
$n/2 < s \Rightarrow S$ is not (contained in) FA or SU

Subnumerals valued between $\sqrt{\mathrm{n}}$ and $n/2$ remain undecidable up to this point. Examples for such yet undecidable subnumerals are abundant. E.g., in English, $N(26) =$ 'twenty-six' has the summand word $SU = N(6)$, but $6^2 > 26$. Hence, it is not yet decidable whether $N(6)$ is a summand or not. Without context, the algorithm cannot exclude that the numeral uses base 6 with $x = 4 * 6 + 2$, so $N(6)$ would be the multiplier. The given counterexample is transferrable to any numeral n in which $su^2 > n = fa * mu + su$. This affects every third numeral as the ratio of pairs $(fa, su) \in \{1, \ldots, mu - 1\}^2$ satisfying $su^2 > n$ is

$$\frac{\#\{(fa,su)\in\{1,\dots,mu-1\}^2\ su^2>n\}}{\#\{1,\dots,mu-1\}^2}=\frac{\sum_{fa=1}^{mu-1}\#\{su\mid\sqrt{fa*mu+su}<su<mu\}}{\#\{1,\dots,mu-1\}^2}$$

$$=\frac{\sum_{fa=1}^{mu-1}\lfloor mu-\sqrt{fa*mu+nu}\rfloor}{(mu-1)^2}\approx\frac{\int_1^{mu}mu-\sqrt{fa*mu}\ d(fa)}{mu^2}\approx\frac{1}{3}$$

For a working decomposer, we should close the gap of decision. We were not able to find a definitive solution. Instead, we use a leaky criterion based on the idea that su is usually not a divisor of x.

Unpacking Criterion 4 (Leaky Criterion). *Let X be a numeral to be interpreted as $fa*mu+su$.*

Let S be a subnumeral of X, such that

- *$N(fa*mu)$ is masked in X,*
- *$x/2>s>\sqrt{x}$*
- *S has no subnumerals.*

Then we assume that S is a subnumeral of SU if and only if $s\nmid fa*mu$.

Justification. $\Leftarrow$: *Given that $s^2>x$, by Unpacking Criterion 3, S cannot be contained in FA. If $s\nmid fa*mu$, then S cannot be MU either. S can be 1)$N(fa'*mu)$ with a subnumeral FA' of FA with $fa'\nmid fa$, or 2)$N(mu+su')$ with a subnumeral SU' of SU, or 3)SU. Any $N(fa'*mu)$ or $N(mu+su')$ is unlikely to be a subnumeral of X given that $N(fa*mu)$ is masked. Also, they likely have subnumerals unlike S does. Hence, we assume that S is a subnumeral of SU.* ◻

$\Rightarrow$: *In this direction, we argue that there is no $x=fa*mu+su$ with $N(fa*mu)$ being masked, SU having no subnumerals, $su^2>x$ and $su|fa*mu$. Although this claim is not generally true, we explain why exceptions are rare and showcase which amount of coincidences they require.*

First, it is rare to have $N(fa*mu)$ masked, especially at higher numbers with 3 digits[4] or more. And, even if a 3-digit numeral had $N(fa*mu)$ masked, it would still require $N(su)$ to have no subnumerals, which often means that su is small, so it is unlikely that $su^2>x$.

For 2-digit numbers, $N(su)$ is 1-digit, so there are few possibilities to construct an exception. In base 10, the only pairs (fa,su)—that satisfy the arithmetic properties $su|fa*mu$ and $su^2>fa*10+su$—are $(1,5)$ and $(4,8)$. So, if in English $N(48)$ would be 'fortaj-eight', while $N(40)$ would still be 'forty', then 'eight' can be suspected as a multiplier, because $8\mid 48$. Base 20 systems offer more space for exceptions. Arithmetically, with $mu=20$ they are possible if $(fa,su)\in$

$$\{(1,10),(2,10),(3,10),(4,10),(3,12),(6,12),(7,14),(3,15),(6,15),(9,15),(4,16),(8,16),(9,18)\}.$$

*Hence, whenever a vigesimal numeral has its subnumeral $N(fa*mu)$ masked, an exception could occur. It actually occurs in French (fr) where the numerals $N(4*20+k)$ for $k=1,\dots,19$ are spelled 'quatre-vingt-$\cdot N(k)$' and do not contain $N(4*20)=$ 'quatre-vingt**s**' letter-by-letter. The numerals $N(4*20+10)=$'quatre-vingt-dix' and $N(4*20+16)=$'quatre-vingt-seize' also fulfill the arithmetic requirements, and $N(10)=$'dix' and $N(16)=$'seize' also do not have subnumerals.*

---

[4] 'Digit' does not necessarily refer to base-10 digits here, but more generally to the coefficients c_i in base-b representation $c_0*b_0+c_1*b_1+\dots$

Posing an exception to Unpacking Criterion 4 leeds to $N(10)$ and $N(16)$ being interpreted as multipliers, as will be seen later. ◻

The use case for Unpacking Criterion 4 may not seem obvious, but in Section 5.3 we present a situation for which it is tailor-made. The leakiness of Unpacking Criterion 4 is not very problematic, as its failure can only cause a small lack in generalization (Objective 1) rather than a problematic overgeneralization (Objective 2).

Comment on working assumption: Generally, it is possible that a numeral N is smaller than its subnumeral. Derzhanski and Veneva (2018) mention two possibilities, namely when N*'s* structure involves overcounting (Dékány, 2025) or subtraction. These possibilities lead to logical violations of the unpacking criteria. However, the unpacking criteria keep their present validity for the main cases $S \in \{FA, MU, SU\}$ and since subtraction and overcounting only allow n' to be slightly larger than n, there are few possibilities for logical violation. Such violation did not practically harm our grammar induction tests.

5. NUMERAL DECOMPOSER ALGORITHM

5.1 Basic Version

In our study, we had first discovered Unpacking Criterion 1, and we noticed that it alone can drive a decent decomposition algorithm. Since it is just a necessary but not sufficient criterion, it may unpack subnumerals larger than FA and SU. Specifically, mu and $mu + su$ are often $< n/2$. In the basic Algorithm 1, we circumvent this issue by setting a checkpoint so that $N(mu)$ and $N(mu + su)$ are never tested on the criterion in the first place.

Algorithm 1 Basic algorithm as pseudocode using Python syntax. cp stands for checkpoint. Function isNumeral returns $True$ iff the input string is a grammatically correct numeral word based on an available lexicon. $n(substring)$ is the number value of the numeral $substring$. Instruction 'Unpack $substring$' adds $substring$ to a list of unpacked subnumerals. A 'Repack...' instruction removes an entry from the list of unpacked subnumerals.

```
1: Decompose numeral
2: cp ← 0
3: for end in range (length(numeral)) do:
4:    for start in range(cp:end) do:
5:       substring ← numeral[start:end]
6:       if substring isNumeral then:
7:          if 2 * n(substring) < n(numeral) then:
8:             Unpack substring
9:             Repack sub-substrings of substring that were unpacked before
10:         else:
11:            cp ← end
12:         end if
13:         break start-loop
14:      end if
15:   end for
16: end for
```

For illustration, we describe the decomposition of the complex English numeral $N(27001) =$ 'twenty seven thousand and one'.

The end- and $start$-loops (lines 3,4) make the code check substrings of 'twenty-seven thousand and one' in the order 't', 'tw', 'w', 'twe', 'we', 'e', 'twen', 'wen'... At $[start\colon end] = [0\colon 6]$, the subnumeral $N(20) =$ 'twenty' is found (ln. 6). Since $20 < 27001/2$ (ln. 7), it gets unpacked (ln. 8). Then the $start$-loop breaks (ln. 13), so the next substring to check is 'twenty-' at $[start\colon end] = [0\colon 7]$ rather than 'wenty' at $[1\colon 6]$. At $[0\colon 12]$, $N(27) =$ 'twenty-seven' is found (ln. 6). Since $27 <$

$27001/2$ (ln. 7), it is also unpacked (ln. 8) and its previously unpacked sub-subnumeral $N(20) =$ 'twenty' is repacked (ln. 9). Moreover, $start$-loop breaks (ln. 13), so the next substring to check is 'twenty seven ' at $[0{:}13]$ and the algorithm will never see the $N(7) =$ 'seven' at $[7{:}12]$. With lines 9 and 13 we make sure that the factor word $N(27)$ is unpacked in one rather than having $N(20)$ and $N(7)$ being unpacked separately. At $[0{:}21]$, $N(27000) =$ 'twenty-seven thousand' is discovered. Since $27000 \nless 27001/2$ (ln. 7,10), it is not unpacked, but the $start$-loop breaks (l. 13), so the algorithm will continue at $end = 22$. This way, it is avoided that the algorithm finds $N(7000)$ at $[7{:}21]$ or $N(mu) = N(1000)$[5] at $[13{:}21]$. If it would find $N(7000)$ or $N(1000)$, it would unpack it, since it is $< 27001/2$. Also, the checkpoint cp is reset to 21 (ln. 11), which makes the $start$-loop no longer check substrings with $start < 21$ (ln. 4). This way, we avoid that $N(7001)$ at $[7{:}29]$ or $N(1001)$ at $[13{:}29]$ may be found and unpacked. Instead, it will find $N(1)$ next at $[26{:}29]$ (ln. 6) and unpack it (ln. 7). The algorithm terminates then (ln. 13) and—as intended—has unpacked the subnumerals $FA = N(27)$ and $SU = N(1)$.

In order to describe such numeral decompositions systematically we use a format as in Decomposition 1. It contains a numeral description with all relevant information about the numeral, including the numeral's language, number value, and desired decomposition and it visualizes **all** subnumerals of the numeral. A zero-based index scale facilitates referencing between $[start{:}end]$ values and substrings. The actual decomposition process is described by a table that explains the algorithm's behaviour at every time when a subnumeral is found (ln. 6).

```
Language: English   Number: 27001 = (20+7)*1000+1
Index:         0 1 2 3 4 5 6 7 8 9 10  12    15         20         25    28
Numeral:       t w e n t y - s e v e n   t h o u s a n d   a n d   o n e
Subnumerals:|---N(20)---| |------------------N(7001)------------------|
            |--------N(27)----------| |------------N(1001)------------|
                          |--N(7)---| |----N(1000)----|          |-N(1)|
            |-----------------N(27000)---------------|
                          |---------N(7000)-----------|
Desired decomposition: _ thousand and _ (27,1)
```

$[start{:}end]$:	$[0{:}6]$	$[0{:}12]$	$[0{:}21]$	$[26{:}29]$
Subnumeral:	$N(20)$	$N(27)$	$N(27000)$	$N(1)$
Criterion:	$< 27001/2$	$< 27001/2$	$\nless 27001/2$	$< 27001/2$
Checkpoint:	0	0	$0 \rightarrow 21$	21
Unpacked:	{20}	{27}	{27}	{27,1}
References:	ln. 7,8	ln. 7-9	ln. 10,11	ln. 7,8

⇒_thousand and _(27,1)

DECOMPOSITION 1: English 'twenty-seven thousand and one' decomposed by basic Algorithm 1.

The example of decomposing 'twenty-seven thousand and one' showcases that the basic algorithm unpacks exactly FA and SU when all of the following 4 conditions are true.

1. The subwords FA, MU and SU are arranged in the order $FA \cdot MU \cdot SU$.
2. $N(fa * mu)$ is not masked. (Otherwise, the checkpoint is not reset properly, so $N(mu + su)$ may be unpacked instead of $N(su)$.)

[5] Officially, $N(1000)$ is 'one thousand' in English. However, we show that it even works if it was just spelled 'thousand'. Same holds for 'thousand and one.

3. $N(mu)$ ends at the same letter as $N(fa * mu)$ or is masked. (Otherwise, $N(mu)$ may be unpacked.)

4. $N(fa)$ and $N(su)$ are not masked. (Otherwise, they cannot be found and unpacked.)

Under these conditions, the algorithm perfectly reverses Hurford's Packing Strategy. Decomposition 2 showcases that the basic version works fine with non-base-10 numerals.

```
Language: Tsez   Number: 86 = 4*20+6
Index:        0        4       8 9       13
Numeral:      u y n o q u n o   i ł n o
Subnumerals:|-N(4)--|-N(20)-| |-N(6)--|
            |----N(80)------|
                    |-----N(26)-------|
Desired decomposition: _ quno _ (4,6)
```

$[start{:}end]$:	$[0{:}4]$	$[0{:}8]$	$[9{:}13]$
Subnumeral:	$N(4)$	$N(80)$	$N(6)$
Criterion:	$< 86/2$	$\geq 86/2$	$< 86/2$
Checkpoint:	0	$0 \to 8$	8
Unpacked:	$\{4\}$	$\{4\}$	$\{4,6\}$
References:	ln. 7,8	ln. 10,11	ln. 7,8

⇒_ quno _(4,6)

DECOMPOSITION 2: Tsez 'uynoquno iłno' decomposed by basic Algorithm 1.

The resulting template '_quno _' is obtained analogously when parsing any other Tsez numeral $N(a * 20 + b)$ for $(a, b) \in \{2,3,4\} \times \{1, \ldots, 19\}$. It constitutes a proper functional equation $(x_1, x_2) \mapsto 20x_1 + x_2$.

In I. Maier and Wolff, 2022 we already published the present basic decomposition algorithm and showed that it is surprisingly well-rounded. However, it still has systematic errors, which we show in the following subsection.

5.2 Problems of the Basic Algorithm

In the last subsection we showed that errors do not appear if the parsed numeral has a generic $FA \cdot MU \cdot SU$ order, MU terminates exactly with MU, and no subnumerals are masked. In this section, we show what errors can occur otherwise and what issues they can cause with respect to the objectives stated in Section 3. This is not a complete analysis of what errors could theoretically appear, but only a summary of what errors we found in our database of 257 languages, see Section 6.1.

Masked subnumerals: As stated in Section 3, masked subnumerals cannot be unpacked and they are not supposed to be unpacked. However, masked subnumerals can also cause a factor or summand not to be unpacked, even if the factor or summand itself is not masked, as the example of Spanish (es) $N(25) =$ 'veinticinco' shows (Decomposition 3). Here, since $FA \cdot MU = N(20) =$ 'veinte' is masked, the checkpoint cp is not moved early enough. The algorithm only enters the if-clause in line 6 for the first time at $[0{:}11]$ with the total numeral $N(25) =$ 'veinticinco'. Nothing is unpacked because $25 > 25/2$ (ln. 7) and the $start$-loop breaks (ln. 13), causing the algorithm not to find $N(5) =$ 'cinco' at all.

```
Language: Spanish   Number: 25 = 20+5
Index:        0           6       10
Numeral:      v e i n t i c i n c o
Subnumerals:              |---N(5)--|
Desired decomposition: veinti_(5)
```

$[start: end]$:	$[0: 11]$	
Subnumeral:	$N(25)$	
Criterion:	$\nless 25/2$	
Checkpoint:	$0 \rightarrow 9$	$\Rightarrow$`veinticinco()`
Unpacked:	{}	
References:	ln. 10,11,13	

DECOMPOSITION 3: Spanish 'veinticinco' decomposed by basic Algorithm 1.

The same issue concerns all Spanish numerals from $N(21)$ to $N(29)$.

We state: **An error can be caused by $N(fa * mu)$ being masked.**

Order of subnumerals: According to Hurford, the subnumerals FA, MU and SU can be in a different order, since the rules in Fig. 1 only represent a hierarchy.

Order $SU \cdot FA \cdot MU$: This order of subnumerals does not generally cause an undue decomposition, as Decomposition 4 showcases.

```
Language: Upper-Sorbian     Number: 61 = 1+6*10
Index:      0          5 6        10         15
Numeral:    j ě d y n a š ě s ć d ź e s a t
Subnumerals:|---N(1)--| |--N(6)-|
                        |-------N(60)-------|
Desired decomposition: _a_ dźesat(1,6)
```

$[start: end]$:	$[0: 5]$	$[6: 10]$	$[0: 16]$	
Subnumeral:	$N(1)$	$N(6)$	$N(61)$	
Criterion:	$< 61/2$	$< 61/2$	$\nless 61/2$	$\Rightarrow$`_a_dźesat(1,6)`
Checkpoint:	0	0	$0 \rightarrow 16$	
Unpacked:	{1}	{1,6}	{1,6}	
References:	ln. 7,8	ln. 7,8	ln. 10,13	

DECOMPOSITION 4: Upper-Sorbian 'jedynašěsćdźesat' decomposed by basic Algorithm 1.

Similar cases occur in Somali, Lower-Sorbian, Slovene, and many Germanic languages. The order $FA \cdot SU \cdot MU$ would be decomposed in the same way, but we have not found any real examples for it in our database.

Order $MU \cdot FA \cdot SU$: This order of subnumerals can cause MU instead of FA to be unpacked as Decomposition 5 showcases.

```
Language: Nyungwe     Number: 34 = 10*3+4
Index:        0   2         7     10      14           20 22
Numeral:      m a k ´ u m i   m a t a t u   n a   z i n a i
Subnumerals:     |--N(10)--|      |--N(3)-|            |-N(4)|
             |-----------N(30)-----------|
Desired decomposition: mak´umi ma_ na zi_(3,4)
```

$[start: end]$:	$[2:7]$	$[0:13]$	$[19:22]$
Subnumeral:	$N(10)$	$N(30)$	$N(4)$
Criterion:	$< 34/2$	$\nless 34/2$	$< 34/2$
Checkpoint:	0	$0 \rightarrow 13$	13
Unpacked:	$\{10\}$	$\{10\}$	$\{10,4\}$
References:	ln. 7,8	ln. 7,8,9,13	ln. 7,8

⇒`ma_ matatu na zi_`(10,4)

DECOMPOSITION 5: Nyungwe 'mak´umi matatu na zinai' decomposed by basic Algorithm 1.

The problem is that subnumeral $N(fa * mu)$—that usually is the first to be $\geq n/2$—is not finished by $N(mu)$ but by $N(fa)$, which lets the basic algorithm confuse FA with MU.

If FA is a compound numeral, the algorithm may only interpret an initial part of FA as MU as Decomposition 6 showcases.

```
Language: Makhuwa     Number: 60 = 10*(5+1)
Index:       0                 9         14       18     21
Numeral:     m i l o k o   m i t h a n u    n a    m o s a
Subnumerals:|-----------N(50)-----------|         |--N(1)-|
                                 |---N(5)--|
                                 |-----------N(6)----------|
Desired decomposition: miloko mi_(6)
```

$[start: end]$:	$[0:14]$	$[18:22]$
Subnumeral:	$N(50)$	$N(1)$
Criterion:	$\nless 60/2$	$< 60/2$
Checkpoint:	$0 \rightarrow 14$	14
Unpacked:	$\{\}$	$\{1\}$
References:	ln. 10,11,13	ln. 7,8

⇒`miloko mithanu na_`(1)

DECOMPOSITION 6: Makhuwa 'miloko mithanu na mosa' decomposed by basic Algorithm 1.

The same issue would arise whenever MU stands before FA, also in orders $MU \cdot SU \cdot FA$ and $SU \cdot MU \cdot FA$. However, we did not find any numerals arranged like this.

We state: An error can be caused if $N(mu)$ appears before $N(fa)$.

We can generalize the statement to: **An error can be caused if** $N(mu)$ ends **before** $N(fa * mu)$. An example is found in Suomi (Finnish, fi) (Decomposition 7).

```
Language: Suomi     Number: 201 = 2*100+1
Index:         0         5       9 10    13
Numeral:       k a k s i s a t a a y k s i
Subnumerals:|---N(2)--|-N(100)| |--N(1)-|
            |-------N(200)------|
Desired decomposition: _sataa_(2,1)
```

$[start\colon end]$:	$[0\colon5]$	$[5\colon9]$	$[0\colon10]$	$[10\colon14]$
Subnumeral:	$N(2)$	$N(100)$	$N(200)$	$N(201)$
Criterion:	$< 201/2$	$< 201/2$	$\nless 34/2$	$< 34/2$
Checkpoint:	0	0	$0 \rightarrow 10$	10
Unpacked:	$\{2\}$	$\{2{,}100\}$	$\{2{,}100\}$	$\{2{,}100{,}1\}$
References:	ln. 7,8	ln. 7,8	ln. 10,11	ln. 7,8

⇒__a_(2,100,1)

DECOMPOSITION 7: Suomi 'kaksisataayksi' decomposed by basic Algorithm 1.

In this case, both $N(fa)$ and $N(mu)$ get unpacked, as neither is right at the end of $N(fa * mu)$.

Overall, we have identified two causes of error:

- **Cause 1: Masked** $FA \cdot MU$ ($N(fa * mu)$ is no subnumeral).

- **Cause 2:** Early MU ($N(mu)$ ends before $N(fa * mu)$ ends)

These causes can lead to two different types of problems:

- Type 1: FA or SU not getting unpacked, despite not being masked.
- Type 2: MU getting unpacked.

Either cause can lead to either type of problem. For each combination, an example numeral word is given in the following table. Each example has been explained in this subsection.

	FA or SU not unpacked	MU unpacked
Masked $FA \cdot MU$	'veinticinco'	'quatre-vingt-seize'
Early MU	'mak´umi matatu na zinai'	'kaksisataayksi'

Problems of type 1 lead to issues with lexicon size (Objective 1). Whenever a FA or SU is not unpacked in a numeral $X = N(fa * mu + su)$, the numeral X cannot be identified with similar numerals like $N(fa' * mu + su)$ or $N(fa * mu + su')$, so X would need its own template.

Problems of type 2 can cause wrong functional equations (Objective 2) because of undue generalization. As mentioned in Section 3, if in English $N(21) - N(29)$ got generalized with $N(27000)$ to a single function twenty-_ with input set $\{1, \ldots, 9, 7000\}$, a correct affine linear functional equation would not exist.

5.3 Advanced Algorithm

In this subsection, we explain how Algorithm 2 solves issues of the basic algorithm. In the following subsections, we show how the added lines 12-32 enhance lexicon size reduction and how the added lines 38-60 avoid overgeneralization.

5.3.1 Dealing with lexicon reduction

In this section we explain lines 1-37 in the advanced Algorithm 2. These code lines are built out of Algorithm 1 and the added lines 12-32. The added lines deal with errors of Algorithm 1 where FA or SA did not get unpacked despite not being masked, such as in the cases of 'veinticinco' and 'mak´umi matatu na zinai'. Fixing these errors enhances desired generalizations of words, whereby FA or SU can be replaced by other factor or summand words. In this way, the lexicon size can be reduced. We present use cases in the following examples.

Algorithm 2 Advanced numeral decomposition algorithm. In the new lines 12-32, further criteria are added under which subnumerals can get unpacked. The new lines 38-60 are tests based on which unpacked multipliers are detected and repacked.

1: Decompose $numeral$
2: $cp \leftarrow 0$
3: **for** end in range(length($numeral$)) **do**:
4: **for** $start1$ in range(cp: end) do:
5: $substring \leftarrow numeral[start1{:}end]$
6: **if** $substring$ isNumeral then:
7: **if** $2 * n(substring) < n(numeral)$ **then**:
8: Unpack $substring$
9: Repack sub-substrings of $substring$ that were unpacked before
10: **else:**
11: $cp \leftarrow end$
12: **for** $start2$ in range($start1 + 1$, end) **do**:
13: $substring2 \leftarrow numeral[start2{:}end]$
14: **if** $substring2$ isNumeral **then**:
15: **if** $n(substring2)^2 \leq n(numeral)$ **then**:
16: Unpack $substring2$
17: Repack sub-substrings of $substring2$
18: $cp \leftarrow start2$
19: $maybeUnpack \leftarrow None$
20: break $start2$-loop
21: **else if** $n(substring2) \nmid n(substring)$ and $n(substring2) * 2 < n(numeral)$ **then**:
22: $maybeUnpack \leftarrow substring2$
23: $maybeCP \leftarrow start2$
24: **else:**
25: $maybeUnpack \leftarrow None$
26: **end if**
27: **end if**
28: **end for**
29: **if** $maybeUnpack \neq None$ **then**:
30: Unpack $maybeUnpack$
31: $cp \leftarrow maybeCP$
32: **end if**
33: **end if**
34: break $start1$-loop
35: **end if**
36: **end for**
37: **end for**
38: $maybeMU$ ← value-largest unpacked subnumeral
39: $otherUnpac \leftarrow$ {unpacked subnumerals} \ $\{maybeMU\}$
40: **if** length($otherUnpac$) = 1 **then**:
41: **if** $n(maybeMU) + n(otherUnpac) = n(numeral)$ **then**:
42: Repack $maybeMU$
43: **else if** $n(maybeMU) * n(otherUnpac) = n(numeral)$ **then**:
44: Repack $maybeMU$
45: **end if**
46: **else if** length($otherUnpac$) = 2 **then**:
47: **if** $n(otherUnpac[0]) * n(maybeMU) + n(otherUnpac[1]) = n(numeral)$ **then**:
48: Repack $maybeMU$
49: **else if** $n(otherUnpac[1]) * n(maybeMU) + n(otherUnpac[0]) = n(numeral)$ **then**:
50: Repack $maybeMU$
51: **end if**
52: **else if** length($otherUnpac$) > 2 **then**:
53: for $unpacked$ in $otherUnpac$ **do**:
54: $maybeFA \leftarrow unpacked$
55: $maybeSUs \leftarrow otherUnpac \setminus \{maybeFA\}$
56: **if** $maybeFA * maybeMU + \Sigma(maybeSUs) = n(numeral)$ **then**:
57: Repack $maybeMU$
58: **end if**
59: **end for**
60: **end if**

Recall Decomposition 5. The factor ’tatu’ did not get unpacked due to early MU (Cause 2). In order to resolve the issue, in line 12, we open a second $start2$-loop that looks for more substrings $substring2$ that end at the current end (ln. 13), so that the factor ’tatu’ can be found at all. In line 15, we check $substring2$ for Unpacking Criterion 2. Since $3^2 \leq 34$, the if clause is entered and $substring2 = N(3) =$ ’tatu’ is unpacked in line 16. In detail, lines 1-37 of Algorithm 2 yield Decomposition 8 for ’mak´umi matatu na zinai’.

$[start: end]$:	$[2:6]$	$[0:13]$	$[9:13]$	$[19:22]$
Subnumeral:	$N(10)$	$N(30)$	$N(3)$	$N(4)$
Criterion:	$< 34/2$	$\nless 34/2$	$\leq \sqrt{34}$	$< 34/2$
Checkpoint:	0	$0 \rightarrow 13$	$13 \rightarrow 9$	9
Unpacked:	$\{10\}$	$\{10\}$	$\{10,3\}$	$\{10,3,4\}$
References:	ln. 7,8	ln.10,11	ln.15,16	ln. 7,8

$\Rightarrow$ma_ ma_ na zi_(10,3,4)

DECOMPOSITION 8: Nyungwe ’mak´umi matatu na zinai’ decomposed by lines 1-37 of advanced Alg. 2.

We will show in Subsection 5.3.2 that $N(10) =$ ’k´umi’ will still get repacked by algorithm lines 38-60. In this way the desired decomposition $34 =$ mak´umi ma_ na zi_$(3, 4)$ will be obtained finally. Note the following details:

1. In line 15, we use the sufficient Unpacking Criterion 2 rather than the necessary Unpacking Criterion 1 to avoid unpacking multiplier words. If we used Criterion 1 instead, errors would appear frequently in basic cases, like in English ’two hundred and one’. When $N(100) =$ ’hundred’[6] is found, Criterion 1 would unpack it, while Criterion 2 does not, since $\sqrt{201} < 100 < 201/2$.
2. In line 18, the checkpoint is reset from the current end to the current $start2$. This becomes important in cases such as the following in which the order is $MU \cdot FA \cdot SU$ and FA is a composed numeral. For a detailed understanding, compare Decomposition 6 with Decomposition 9.

$[start: end]$:	$[0:14]$	$[9:14]$	$[9:22]$
Subnumeral:	$N(50)$	$N(5)$	$N(6)$
Criterion:	$\nless 60/2$	$\leq \sqrt{60}$	$< 60/2$
Checkpoint:	$0 \rightarrow 14$	$14 \rightarrow 9$	9
Unpacked:	$\{\}$	$\{5\}$	$\{6\}$
References:	ln. 10	ln. 15,16,18	ln. 7,8,9

$\Rightarrow$miloko mi_(6)

DECOMPOSITION 9: Makhuwa ’miloko mithanu na mosa’ decomposed by advanced Algorithm 2.

Cause 1 (Masked $N(fa * mu)$) has also caused summand words not to get unpacked by Algorithm 1, such as in ’veinticinco’. In the case of ’veinticinco’, this issue is already resolved with lines 15-20 of Algorithm 2 (see Decomposition 10).

$[start: end]$:	$[0:11]$	$[6:11]$
Subnumeral:	$N(25)$	$N(5)$
Criterion:	$\nless 25/2$	$\leq \sqrt{25}$
Checkpoint:	$0 \rightarrow 11$	$11 \rightarrow 6$
Unpacked:	$\{\}$	$\{5\}$
References:	ln. 10	ln. 15,16

$\Rightarrow$veinti_(5)

DECOMPOSITION 10: Spanish ’veinticinco’ decomposed by advanced Algorithm 2.

[6] Here we assume that $N(100) =$ ’hundred’ rather than ’one hundred’. Otherwise, the example works similar with $N(201)$ in Deutsch or various other languages.

Correct decompositions of $N(21)$ = 'veintiuno', ..., $N(24)$ = 'veintiquatro' are obtained analogously. However, for $N(20+x)$ with $x>5$, the summand x is larger than $\sqrt{20+x}$, so it does not satisfy Criterion 2. So, when processing 'veintiseis' with Algorithm 2, lines 15-20 do not cause $N(6)$ = 'seis' to be unpacked. Therefore, we added the else-if clause in line 21, to deal with numerals of order $FA \cdot MU \cdot SU$ with masked $N(fa * mu)$.

Usually, $N(fa * mu)$ is the substring that enters the else-clause in line 10, since it is $> n/2$. However, when $N(fa * mu)$ is masked, it never becomes $substring$. Instead, another substring, at latest the total $N = N(fa * mu + su)$ itself, will eventually be $> n/2$. Precisely, it will be $substring = FA \cdot MU \cdot SU'$, in which SU' is the minimal suffix of SU that makes $FA \cdot MU \cdot SU' = N(fa * mu + su')$ a proper numeral word. Then, after $FA \cdot MU \cdot SU'$ has been processed (ln. 6,7,10), a $substring2$ is browsed for. While the if-clause in line 15 checks Unpacking Criterion 2, the else-if-clause in line 21 checks the remaining arithmetic requirements of Criterion 4 for $S = substring2$. Note that the requirement $n(substring2)|fa * mu$ is equivalent to $n(substring2)|n(substring)$, since $substring = N(fa * mu + substring2)$. In order to use Unpacking Criterion 4, it is still required that $substring2$ has no subnumerals. Therefore, $substring2$ is not immediately unpacked after passing line 21. Instead, it is saved as $maybeUnpack$ (ln. 22). If and only if another $subnumeral2$ is found inside $maybeUnpack$ at a later $start2$ (ln. 14), $maybeUnpack$ is reset again (ln. 19, ln. 22 or ln. 25). If not, then we assume that $maybeUnpack = SU'$ has no subnumerals and unpack it in line 30, as it fulfills the requirements of Criterion 4. In this way, the Spanish numeral $N(26)$ = 'veintiseis' gets decomposed properly by Algorithm 2 (Decomposition 11).

```
Language: Spanish     Number: 26 = 20+6
Index:        0           6     9
Numeral:      v e i n t i s e i s
Subnumerals:            |--N(6)-|
Desired decomposition: veinti_ (6)
```

$[start:end]$:	$[0:10]$	$[6:10]$	
Subnumeral:	$N(26)$	$N(6)$	
Criterion:	$< 26/2$	$< 26/2 \wedge \mid 26 \wedge$ *is atom*	⇒veinti_(6)
Checkpoint:	$0 \rightarrow 10$	$10 \rightarrow 6$	
Unpacked:	{}	{6}	
References:	ln. 10	ln. 21,22,29,30	

DECOMPOSITION 11: Spanish 'veintiseis' decomposed by advanced Algorithm 2.

It works the same for $N(27) - N(29)$. Also, for most French numerals of the shape 'quatre-vingt-'$\cdot X$, the summand X now gets unpacked properly (see Decomposition 12).

However, since Criterion 4 is leaky, X can still remain packed in rare cases like Decomposition 13 in which $n(X)$ | n('quatre-vingt-'$\cdot X$).

The same issue also appears for 'quatre-vingt-dix', which is decomposed _-_-dix$(4, 20)$. A related issue appears in Decomposition 14 with the numerals $N(97)$, $N(98)$ and $N(99)$, as their summand begins with the sub-subnumeral SU' = 'dix' = $N(10)$ and $n(SU')|n$('quatre-vingt-'$\cdot SU'$).

This far we have attained proper unpacking of summand words. In the few cases in French in which summands are not unpacked, some lexicon efficiency is lost due to the leakiness of Unpacking Criterion 4. Specifically, the five numerals $N(90)$ and $N(96) - N(99)$ cannot be covered by the function _-vingt-_, but need their own separate lexicon entries.

```
Language: French      Number: 91 = 4*20+11
Index:        0           6           12       16
Numeral:     q u a t r e - v i n g t - o n z e
Subnumerals:|----N(4)---| |--N(20)--| |-N(16)-|
Desired decomposition: _ -vingt-_ (4,11)
```

$[start: end]$:	$[0: 6]$	$[7: 12]$	$[0: 17]$	$[13: 17]$
Subnumeral:	$N(4)$	$N(20)$	$N(91)$	$N(11)$
Criterion:	$< 91/2$	$< 91/2$	$\nless 91/2$	$\nless 91/2 \wedge$ | 91 ∧is atom
Checkpoint:	0	0	$0 \rightarrow 17$	$17 \rightarrow 12$
Unpacked:	$\{4\}$	$\{4,20\}$	$\{4,20\}$	$\{4,20,11\}$
References:	ln. 7,8	ln. 7,8	ln. 10	ln. 21,22,29,30

⇒_-_-_(4,20,11)

DECOMPOSITION 12: French 'quatre-vingt-onze' decomposed by ln. 1-37 of advanced Alg. 2.

```
Language: French      Number: 96 = 4*20+16
Index:        0           6           12        17
Numeral:     q u a t r e - v i n g t - s e i z e
Subnumerals:|----N(4)---| |--N(20)--| |--N(16)--|
Desired decomposition: _- vingt-_ (4,16)
```

$[start: end]$:	$[0: 6]$	$[7: 12]$	$[0: 18]$	$[13: 18]$
Subnumeral:	$N(4)$	$N(20)$	$N(96)$	$N(16)$
Criterion:	$< 96/2$	$< 96/2$	$\nless 96/2$	$\nless \sqrt{96}$ and |96
Checkpoint:	0	0	$0 \rightarrow 18$	18
Unpacked:	$\{4\}$	$\{4,20\}$	$\{4,20\}$	$\{4,20\}$
References:	ln. 7,8	ln. 7,8	ln. 10	ln. 24

⇒_-_-seize(4,20)

DECOMPOSITION 13: French 'quatre-vingt-seize' decomposed by advanced Algorithm 2.

```
Language: French      Number: 99 = 4*20+19
Index:        0           6           12      16      20
Numeral:     q u a t r e - v i n g t - d i x - n e u f
Subnumerals:|----N(4)---| |--N(20)--| |-----N(19)-----|
            |--------------N(90)-------------| |--N(9)-|
                                      |N(10)|
Desired decomposition: _-vingt-_(4,19)
```

$[start: end]$:	$[0: 6]$	$[7: 12]$	$[0: 16]$	$[13: 16]$	$[17: 21]$
Subnumeral:	$N(4)$	$N(20)$	$N(90)$	$N(10)$	$N(9)$
Criterion:	$< 99/2$	$< 99/2$	$\nless 99/2$	$\nleq \sqrt{99}$ and |90	$< 99/2$
Checkpoint:	0	0	$0 \rightarrow 16$	16	16
Unpacked:	$\{4\}$	$\{4,20\}$	$\{4,20\}$	$\{4,20\}$	$\{4,20,9\}$
References:	ln. 7,8	ln. 7,8	ln. 10	ln. 24	ln. 7,8

⇒_-_-dix-_(4,20,9)

DECOMPOSITION 14: French 'quatre-vingt-dix-neuf' decomposed by advanced Algorithm 2.

5.3.2 Dealing with Overgeneralization

As can be seen in the examples of ’mak´umi matatu na zinai’, ’kaksisataayksi’ and ’quatre-vingtdix’, whenever $N(mu)$ ends before $N(fa * mu)$ (like ’k´umi’ in ’mak´umi matatu’, ’sata’ in ’kaksisataa’ and ’vingt’ in ’quatre-vingts’), then Algorithm 1 unpacks the multiplier despite our intention. This leads to decompositions like $80 =$ _-_s$(4{,}20)$ or $201 =$ _ _a_$(2{,}100)$. Such functions are prone to overgeneralization. Specifically, ’_ _a_’ cannot only generate Suomi numerals between 10^2 and 10^3, but also numerals between 10^6 and 10^{12} when the second input is $N(10^6) =$’miljoona’ or $N(10^9) =$ ’miljardi’ instead of $N(100) =$ ’sata’. This is an undue overgeneralization with respect to Objective 2, since ’_ _a_’ cannot work with an affine linear function in this value range. Algorithm 2 overcomes this issue with the added lines 38-60. There, it looks for clues to detect an unpacked MU in order to repack it.

If a numeral X has three unpacked subnumerals U_1, U_2, U_3 that happen to satisfy $u_1 * u_2 + u_3 = x$ with $u_1 < u_2$, then we suspect $MU = U_2,\ FA = U_1$ and $SU = U_3$, hence we would repack U_2. Note that first one would need to find the distribution of the unpacked subnumerals on the roles $FA,\ MU$ and SU. In light of Axioms 2 and 3, MU would always have the largest value of all.

Thus, if we have three unpacked subnumerals, our strategy is: Set the value-largest unpacked subnumeral U_{max} to $maybeMU$ (ln. 38), as we suspect it may be MU. If the other two U_1, U_2 satisfy $u_1 * u_{max} + u_2 = x$ (ln. 47) or $u_2 * u_{max} + u_1 = x$ (ln. 49), then we repack $maybeMU = U_{max}$ (ln. 48,50), as the suspicion $U_{max} = MU$ has been strengthened.

If we have a number of unpacked subnumerals different from three, we use a similar strategy, as can be seen in lines 40-45 and 52-59 of Algorithm 2.

With this fix, we were able to solve the errors of type MU unpacked on a large scale.

For example, it solved the issue in the decomposition of ’kaksisataayksi’ $= 201 =$ _ _a_$(2, 100, 1)$, as the algorithm can find out that $2 * 100 + 1 = 201$ and detect 100 as a multiplier. By repacking $N(100)$, the desired decomposition $201 =$ _ sataa_$(2, 1)$ is obtained. Likewise, it works in the cases of ’quatre-vingts’ and ’mak´umi matatu na zinai’.

Numeral	$N(80) =$ quatre-vingt	$N(34) =$ mak´umi matatu na zinai
	$\Downarrow$	$\Downarrow$
Alg. 2 dec. till l. 37	_-_s$(4, 20)$	ma_ ma_ na zi_$(10, 3, 4)$
	$\Downarrow$	$\Downarrow$
Finding	$80 = 4 * 20$	$34 = 10 * 3 + 4$
	$\Downarrow$	$\Downarrow$
Diagnosis	$N(20)$ is MU	$N(10)$ is MU
	$\Downarrow$	$\Downarrow$
Alg. 2 final dec.	_-vingts(4)	mak´umi ma_ na zi_$(3, 4)$

If, in the processing of ’quatre-vingt-seize’, Algorithm 2 would have unpacked the summand $N(16) =$ ’seize’ properly, then it would also repack the multiplier $N(20) =$ ’vingt’ after noticing that $96 = 4 * 20 + 16$. This would have led to the desired decomposition $96 =$ _-vingt-_$(4, 16)$. Since ’seize’ did not get unpacked, the algorithm only checks whether or not $96 = 4 * 20$ or $96 = 4 + 20$ and does not notice that $96 = 4 * 20 + 16$. By lacking this clue, ’vingt’ remains unpacked despite our intention. The same happened for ’quatre-vingt-dix’ and the ’quatre-vingt-dix’$\cdot N(y)$ for $y \in \{7, 8, 9\}$. However, the French numerals $N(81) - N(89)$ and $N(91) - N(96)$ are properly decomposed by Algorithm 2 as _-vingt-_$(4,$ _$)$.

While many errors got fixed by lines 38-60, a few new ones were caused, such as in Decomposition 15.

```
Language: Sakha     Number: 299 = 2*100+99
Index:       0       4         9           15      18         23
Numeral:     и к к и   с ү ү с   т о ҕ у с   у о н   т о ҕ у с
Subnumerals:|--N(2)-| |-N(100)| |---N(9)--| |N(10)| |---N(9)--|
            |------N(200)-----| |------N(90)------|       |N(3|
                          |N(3|       |N(3| |------N(19)------|
            |------------N(209)-----------|
                      |-------N(109)------|
            |----------------N(290)---------------|
                      |-----------N(190)----------|
                                |------N(90)------|
                                      |---N(30)---|
                      |-----------------N(199)----------------|
                                |------------N(99)------------|
                                      |---------N(39)---------|
Desired decomposition: _ сүүс _(2,99)
```

$[start: end]$:	$[0:4]$	$[0:9]$	$[5:9]$	$[7:9]$	$[10:15]$	...	$[10:25]$
Subnumeral:	$N(2)$	$N(200)$	$N(100)$	$N(3)$	$N(9)$		$N(99)$
Criterion:	$< \frac{299}{2}$	$\nless \frac{299}{2}$	$\nleq \sqrt{299}$	$\leq \sqrt{299}$	$< \frac{299}{2}$		$< \frac{299}{2}$
Checkpoint:	0	$0 \rightarrow 9$	9	$9 \rightarrow 7$	7	7	7
Unpacked:	{2}	{2}	{2}	{2,3}	{2,3,9}	...	{2,3,99}
References:	7,8	10,11	24	15,16,18	7,8	7,8,9	7,8,9

⇒_ сү_ _(2,3,99)

⇒ Diagnosis: $299 = 3 * 99 + 2$

⇒ $N(99)$ is MU

⇒_ сү_тоҕус уон тоҕус(2,3)

DECOMPOSITION 15: Sakha 'икки сүүс тоҕус уон тоҕус' decomposed by advanced Algorithm 2.

The Sakha (Yakut) numeral $N(100)$ = 'сүүс' accidentally contains $N(3)$ = 'үс', hence, in $N(299)$, it gets unpacked among $N(2)$ and $N(99)$ due to its small value. Since $3 * 99 + 2 = 299$, there is the suspicion that $N(99)$ is a multiplier, so it is repacked and the final decomposition is $299 =$ _ сү_ тоҕус уон тоҕус$(2, 3)$. Similar errors happen in 4 other languages: Breton, Rapa-Nui, Tok-Pisin, and Lachixio-Zapotec. Note that these errors do only minor damage, as they only require one extra lexicon entry for the single incorrectly decomposed numeral.

6. EVALUATION

We evaluated Algorithm 2 by testing it on data sets of numerals and analyzing the produced output lexica. The data sets are described in Subsection 6.1. The data set of English numerals < 1000 induced the following lexicon of template functions:

one:	$() \mapsto 1$	two:	$() \mapsto 2$	three:	$() \mapsto 3$
four:	$() \mapsto 4$	five:	$() \mapsto 5$	six:	$() \mapsto 6$
seven:	$() \mapsto 7$	eight:	$() \mapsto 8$	nine:	$() \mapsto 9$
ten:	$() \mapsto 10$	eleven:	$() \mapsto 11$	twelve:	$() \mapsto 12$
thirteen:	$() \mapsto 13$	_teen:	$(x) \mapsto x + 10$	fifteen:	$() \mapsto 15$
-een:	$() \mapsto 18$	twenty:	$() \mapsto 20$	twenty - _:	$(x) \mapsto x + 20$
thirty:	$() \mapsto 30$	forty:	$() \mapsto 40$	forty - _:	$(x) \mapsto x + 40$
thirty-_:	$(x) \mapsto x + 30$	_ty:	$(x) \mapsto 10 * x$	_ty -_:	$(x, y) \mapsto 10 * x + y$
fifty:	$() \mapsto 50$	_y:	$(x) \mapsto 80$	_y -_:	$(x, y) \mapsto 10 * x + y$
fifty - _:	$(x) \mapsto x + 50$	_hundred:	$(x) \mapsto 100 * x$	_hundred and _:	$(x, y) \mapsto 100 * x + y$

We have left out the domains of the functions to save space. All functional equations are correct (Objective 2) and the lexicon comprises only 30 templates (Objective 1). It resembles expert-made grammars, as it is morphologically plausible. A broader and more comparative evaluation follows in the upcoming subsections. In Subsection 6.2 we compare three induced grammars with expert-made gold standards. In Subsection 6.3 we analyze the correctness of all induced grammars (Objective 2), and in Subsection 6.4 the compactness (Objective 1). In Subsection 6.5 we attempt to present overall error statistics.

6.1 Data

From languagesandnumbers.com we obtained dictionaries of number-numeral pairs for numbers up to 999 in 242 languages, unless a language does not deliver numerals up to 999.

From the Python package num2words, we got a dictionary of number-numeral pairs for numbers up to 1000 and a sample of 4-digit and 5-digit numbers in 35 languages. The sample contains the number 27206 and all 4- and 5-digit numbers that we could reasonably imagine to be contained in $N(27206)$ as a subnumeral in a base-10 system, which are

$$1002, 1006, 1100, 1200, 1206, 7000, 7002, 7006, 7100, 7200, 7206, 10000, 17000,$$
$$17200, 17206, 20000, 27000, 27006 \text{ and } 27200. \quad (3)$$

In base 20 or other base *X* systems, other subnumerals would be conceivable, but all base-20 system languages that we have in our database either transition into base 10 when numbers become bigger, or the database does not have numerals for numbers over 1000. 13 of the languages from num2words are not obtained from languagesandnumbers.com.

Using the TeX code from Derzhanski and Veneva (2020), we generated Birom numerals till 120 and Yoruba numerals till 184. These 2 languages are not included in the other sources.

All data sets only contain the standard grammatical forms of the numerals, since we assume that any challenge that an alternate form may pose on the performance of the numeral decomposer comes up in an analogous form in another language. [7] In Appendix A, all data sets are listed.

6.2 Comparison with Expert-Made Grammars

In this subsection, we compare three expert-made grammars with their numeral-decomposer-induced counterparts. Derzhanski and Veneva (2020) present TeX implementations of grammars for Bulgarian numerals till 99, Birom numerals till 120 and Yoruba numerals till 184. The grammars come from solutions of exercises of the International Linguistic Olympiad and other linguistic contests. The authors chose Bulgarian, Birom, and Yoruba because their numeral systems offer a great variety of features. In particular, Bulgarian uses a standard base-10 system with subnumeral order factor-multiplier-summand, Birom uses a base-12 system involving backward counting and order multiplier-factor-summand, and Yoruba uses a combination of base 20 and base 10 with even more backward counting and order summand-multiplier-factor.

Table 1 shows a direct comparison of induced and expert-made grammar for Birom.

Comparisons for Bulgarian and Yoruba can be found in Appendix B. Notably, for Bulgarian both numeral decomposer version induced the same grammar as the experts, so they worked perfectly.

In order to evaluate the other comparisons, we calculate the accuracy value that Hammarström (2008) used. He interprets grammars as clusters, with each cluster representing the set of expressions generated by a single rule. Accuracy is defined as

[7] The interested reader may challenge this claim by testing the decomposer published on GitHub, see I. K. Maier, 2023.

$$Acc = \frac{\frac{1}{|I|}\sum_{r\in I} prec(r,G) + \frac{1}{|G|}\sum_{r\in G} prec(r,I)}{2} \text{ with}$$

$$prec(r,X) = \frac{|r| - |\{r_x \in X \mid r \cap r_x \neq \emptyset\}| + 1}{|r|}.$$

The formula is similar to cluster purity, see Manning et al., 2008, chapter 16.3. While purity measures how much of one induced rule can be covered by one gold rule, accuracy measures how many gold rules it takes to cover one induced rule completely.

Induced grammar		Expert-made grammar	
Rule/Function	Values	Rule/Function	Values
ATOMS	$1,\ldots,8,12$	ATOMS	$1,\ldots,8,12$
Sāā_ $(x) \mapsto -1x+12$	$9,10,11$	Sāā_ $(x) \mapsto -1x+12$	$9,10,11$
bākūrū bī_ $(x) \mapsto 12x$	$12x$ for $x \in \{2,\ldots,8\}$	bākūrū bī_ $(x) \mapsto 12x$	$12x$ for $x \in \{2,\ldots,8\}$
bā_ Sāābī_ $(x,y) \mapsto 12x - 12y$	$180, 120$	bā_ Sāābī_ $(x,y) \mapsto 12x - 12y$	$180, 120$
kūrū na gwĒ _ $(x) \mapsto 13$	13	_ na gwĒ _ $(x,y) \mapsto x+y$	$12x'+1$ for $x' \in \{1,\ldots,9\}$
bākūrū bī_ na gwĒ _ $(x,y) \mapsto 12x+y$	$12x+1$ for $x \in \{2,\ldots,8\}$		
bā_ Sāābī_ na gwĒ _ $(x,y,z) \mapsto 109$	109		
kūrū na vE_ $(x) \mapsto x+12$	$14,\ldots,23$	_ na vE_ $(x,y) \mapsto x+y$	$12x'+y$ for $x' \in \{1,\ldots,9\}$ $y \in \{2,\ldots,11\}$
bākūrū bī_ na vE_ $(x,y) \mapsto 12x+y$	$12x+y$ for $x \in \{2,\ldots,8\}$ $y \in \{2,\ldots,11\}$		
bā_ Sāābī_ na vE_ $(x,y,z) \mapsto 8x+4y+z$	$110,\ldots,119$		

TABLE 1: An advanced-numeral-decomposer induced grammar for Birom language in comparison with an expert-made gold standard. The decomposer has about the same idea as the expert but it splits up '_na gwĒ _' and '_ na vE_' in three functions to avoid overgeneralization.

Accuracies	Bulgarian	Birom	Yoruba
Advanced Numeral Decomposer	100%	99.13%	98.73%
Basic Numeral Decomposer	100%	89.28%	89.06%

TABLE 2: Accuracy values (Hammarström, 2008) of grammars induced by basic and advanced numeral decomposer in relation to Derzhanski and Veneva (2020)'s expert-made grammars. For Bulgarian, both decomposer versions induced the same grammar as the experts made.

For the Birom induced grammar, we calculate an accuracy of $99.13\,\%$. All induced rules can be covered with one single expert-made rule, so $prec(r,G) = 1$ for all $r \in I$. Out of the $r \in G$, the 9 atoms and the 3 functions 'Sāā_', 'bākūrū bī_' and 'bā_ Sāābī_' are covered by one single induced rule each, while '_ na gwĒ _' and '_ na vE_' are covered by 3 rules. Thus

$$Acc_{Birom} = \frac{1 + \frac{1}{14}\left(12.1 + \frac{9-3+1}{9} + \frac{90-3+1}{90}\right)}{2} = 0.9913.$$

The accuracies of the other induced grammars can be found in Table 2. For comparison, the numeral grammars induced by Hammarström (2008) had an average accuracy of $71.56\,\%$ and amedian accuracy of $90\,\%$.

6.3 Regarding Correct Functional Equations

For the induced grammars, the functional equation of each template is calculated by affine linear regression. A functional equation of a template is correct if it computes the correct number value for each numeral generated by the template. This subsection reports on all incorrect functional equations found in our data.

We note that apart from our data, which do not go beyond 10^6, big English numeral words like 'trillion', 'quadrillion', 'quintillion' etc. are related to the Latin numerals 'tria', 'quattuor', 'quinque', while the impact of these implied subnumerals is not linear but exponential.

Inside our data, we have summarized a report regarding Objective 2 in the following table. Not all errors are caused by bad decomposition. Some occurred due to unintuitive context sensitivity, which we will explain later.

Error causes	Languages
Bad decomposition	3: Tongan, Kiribati, Nyungwe
Context sensitivity	4: Choapan-Zapotec, Nume, Farsi (Persian), Hebrew (he)
Incorrect input data(?)	7: Haida, Purepecha, Susu, Dogrib, Tunica, Yao, Yupi
No errors	243: the rest

In 243 out of 257 languages, the advanced numeral decomposer did not do any undue generalizations. So, for each template function in these 243 languages, an affine linear equation was found that interprets all its output numerals with the correct number value.

In the 14 remaining languages, undue generalizations led to inexact functional equations and thus incorrect interpretations of numerals.

In 11 out of the 14 failed languages, we consider the wrong interpretation reasonable enough that humans could misinterpret them as well.

In many of these cases, we suspect that the data from languagesandnumbers.com have errors: In 6 languages, Purepecha, Susu, Dogrib, Tunica, Yao, and Yupik, we found pairs of numbers with exact same numeral. It is also concievable that these pairs actually differ in intonation or something, and the differences are just not visible in the delivered written form. We also suspect wrong data in Haida, which we explain later.

In the other 5 languages out of the 11, context sensitivities led to errors, i.e., there are compound numerals $X \cdot Y$ and $X' \cdot Y'$, in which (X, X') and (Y, Y') are pairs of intuitively similar numerals but the calculation of $n(X \cdot Y)$ out of x and y is fundamentally different from the calculation of $n(X' \cdot Y')$ out of x' and y':

Choapan-Zapotec: While $N(1) =$ 'tu', $N(2) =$ 'chopa' and $N(3) =$ 'tzona', and 'chopa galo' and 'tzona galo' mean $2 * 20$ and $3 * 20$, respectively, the numeral 'tu galo' means $20 - 1$ instead of $1 * 20$.

Nume: When a 1-digit numeral S (in base 10) is affixed to 'muweldul ', then it means $100 * s$, but if S is a 2-digit numeral, then is means $100 + s$.

Farsi (Persian): In the Latin-transcripted form, we have $N(600) =$'sheshsad', composed as $N(6)\,\cdot$'sad'. The numeral $N(300)$ is similarly composed, but $N(3) =$'se' gets inflected to 'si', which accidentally is $N(30)$, so we have a template _sad mapping 6 to 600 and 30 to 300. Actually, an affine linear equation $x \mapsto 600 + \frac{300-600}{30-6} * (x - 6)$ is still construable, but for code efficiency reasons we have only allowed integer coefficients for the functional equations.

Haida: While $N(2) =$ 'sdáng', $N(3) =$ 'hlgúnahl' and $N(8) =$ 'sdáansaangaa', and 'lagwa uu sdáng' and 'lagwa uu hlgúnahl' mean $2 * 20$ and $3 * 20$, respectively, according to languages and numbers.com the numeral 'lagwa uu sdáansaangaa' means 80 instead of $8 * 20$. We suspect that this is wrong information, since according to omniglot.com, 80 means 'lagwa uu stánsang', which is logical since 'stánsang' $= N(4)$.

Hebrew: While $N(3) =$ 'שלוש', $N(4) =$ 'ארבע' and $N(10) =$ 'עשר', and $3 * 10$ and $4 * 10$ are written 'שלושים' and 'ארבעים', respectively, the numeral 'עשרים' means $10 + 10$ instead of $10 * 10$. In the 3 remaining languages, undue generalizations were made due to bad decompositions:

In Tongan-Telephone-Style, numbers are—with some minor inflections—simply called by the sequence of their decimal digits. The total lack of multiplier words leads to various words being identified with the template _ _. This template can sometimes mean $(x_0, x_1) \mapsto 10 * x_0 + x_1$ for 2-digit numerals and sometimes $(x_0, x_1) \mapsto 100 * x_0 + x_1$. The fact that the rough value size of a numeral cannot be instantaneously estimated during reading—as any further number of digits could still be added— also makes it impossible to make proper use of the separated $start2$-loop in the algorithm. One could argue that this language does not really follow Hurford's Theory of Numerals. This can be justified by an argument that the development of this numeral language is more influenced by telecommunication technology than by nature, so those numerals may not be considered a natural part of a language.

In Gilbertese (Kiribati), the numerals $N(90)$ and $N(900)$ can accidentally be presented as ru $\cdot N(40)$ and ru $\cdot N(400)$. This causes the numerals $N(90 + s)$ and $N(900 + s)$ to be decomposed ru_$(40 + s)$ and ru_$(400 + s)$, respectively. A unification of these templates ru_ has no proper affine linear equation, since the points $(41, 91), (42, 92), (401, 901)$ do not lie on a straight line.

In Nyungwe, again multipliers got unpacked and generalized. The numerals $N(31), N(41), N(301)$, and $N(401)$ got all identified with the template ma_ ma_ na ibodzi with the inputs $(10, 3), (10, 4), (100, 3)$, and $(100, 4)$, respectively. As these input-output combinations do not lie on a straight surface, an affine linear functional equation for the template ma_ ma_ na ibodzi does not exist.

6.4 Regarding Lexicon Sizes

In this subsection, we discuss the lexicon sizes of numeral-decomposer induced grammars, which according to Objective 1 should be as small as possible. Lexicons containing undue generalization, as reported in Subsection 5.2, are not excluded. Some data sets have been removed from the analysis to avoid having two data sets for one language.

Fig. 3 shows how many different template functions the two numeral decomposer versions induce to cover the numerals from 1 to 1000 plus the sample of 4 to 5-digit numbers (Eq. 3) in 34 languages. The languages are sorted by the y values of the advanced version, so one can conclude that, e.g., in 24 of the 34 languages, the advanced numeral decomposer covers the numerals in 50 templates or less. Fig. 4 shows the number of induced templates for numerals up to 999 and 399, respectively. For these ranges of values, we have data from over 200 languages. Therefore, the x axis does not show the names of the languages, but it represents their ordinal positions with respect to the number of advanced-numeral-decomposer induced template functions. The plots imply that the advanced numeral decomposer induces compact numeral grammars in most languages. In 168 out of 202 languages, it maps the numerals till 999 to 50 templates or less. In only 8 languages, it produces over 100 different templates, which are Bavarian, Makhuwa, Hayastani (Armenian), Kartvelian (Georgian), Zulu, Timbisha, Xhosa and Kannada (kn). These languages have in common that most subnumerals of compound numerals are masked due to dropping or inflecting their last or first letter(s).

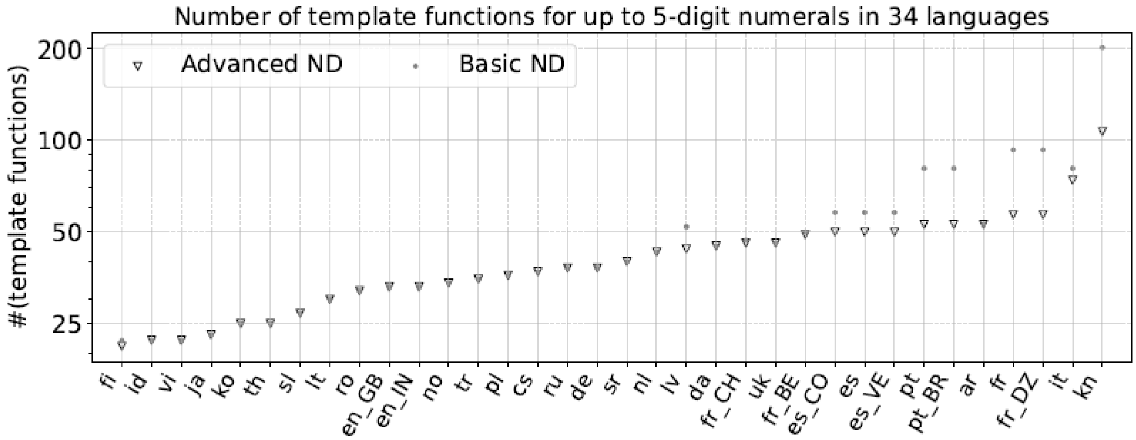


FIGURE 3: Sizes of grammars induced by numeral decomposer versions for numerals of numbers $1 - 1000$ and the numbers in Equation 3.

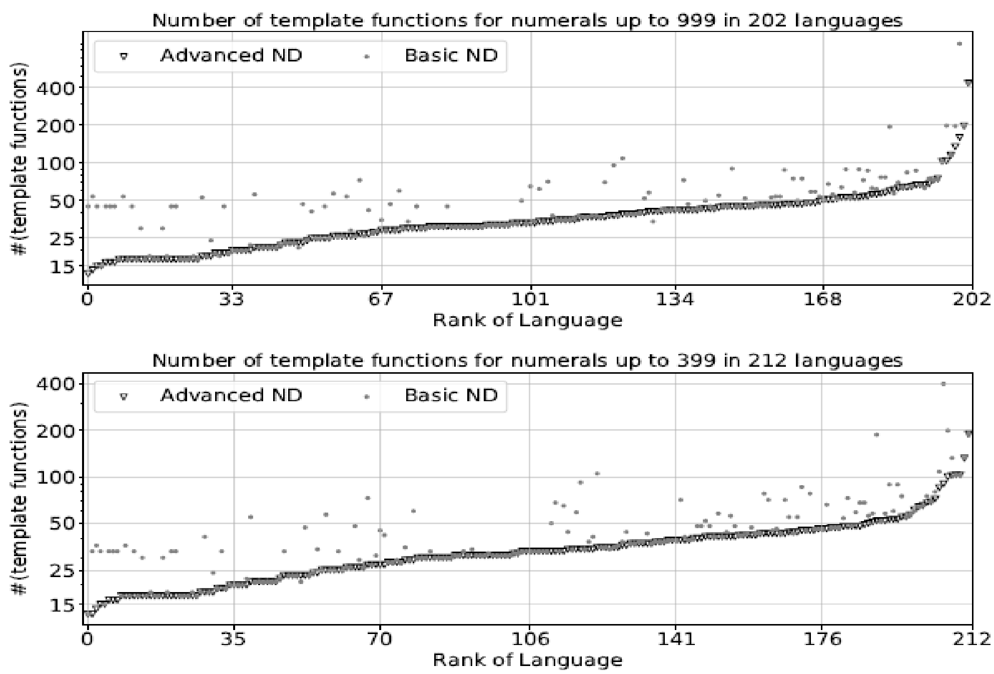


FIGURE 4: The black data points show in how many languages the advanced numeral decomposer induced less than 100, 50 or 25 different template functions to cover numerals till 999 (top) or till 399 (bottom). Only about a sixth of the languages got more than 50 templates to get their numerals till 999 covered. The grey data points show that the basic decomposer induces much larger grammars in some languages.

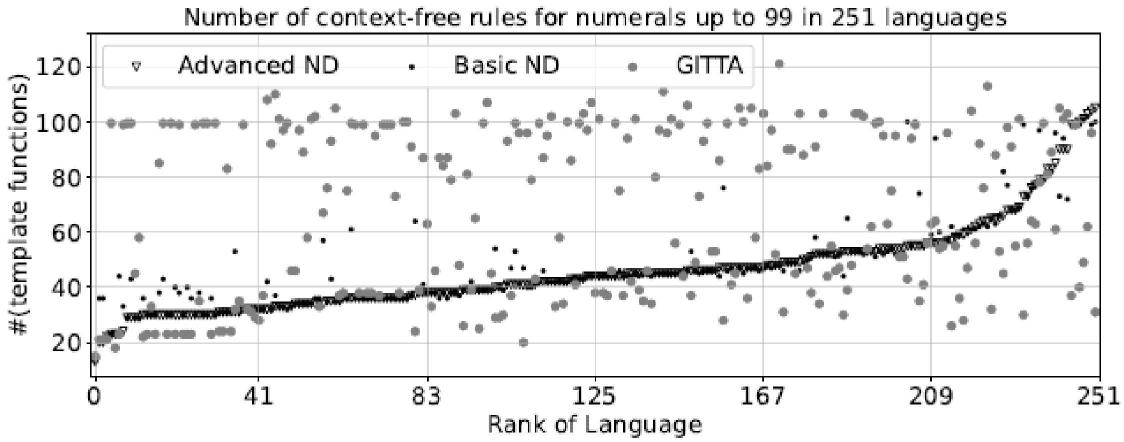


FIGURE 5: Context-free-grammar induction by numeral decomposers and by GITTA. In about two thirds of the languages the advanced numeral decomposer induces exact CFGs for numerals till 99 with less than 50 rules. GITTA's CFGs are mostly larger.

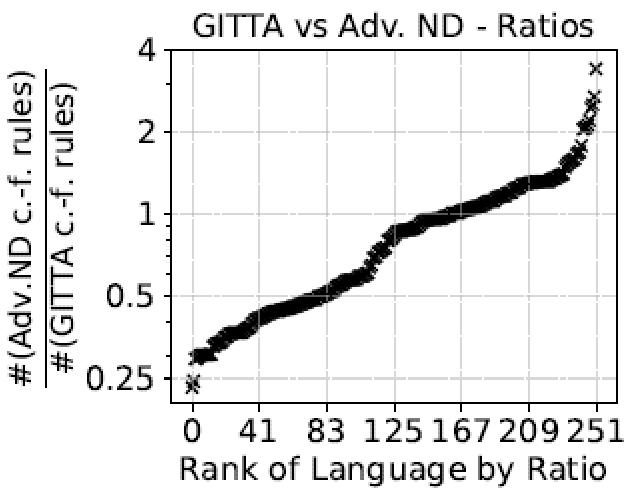


FIGURE 6: In about two thirds of languages the CFGs induced by the advanced decomposer are smaller than GITTA's CFGs. In about a third of the languages they are smaller than half of GITTA's size.

The scattered grey data points above the black curves in Figures 3 and 4 show that the templates induced by the basic numeral decomposer are occasionally less generalizing than the advanced numeral decomposer's induced templates.

In order to give a comparison, we conducted grammar induction for numerals till 99 not only with the numeral decomposer versions but also with GITTA (Winters & Raedt, 2020). GITTA is a general tool that induces context-free grammars for natural language input. We assist GITTA by adding spaces into the numerals at any position where a subnumeral begins or ends.

To make the comparison fair, we have to convert the numeral template grammars into CFGs. Therefore, each template function f becomes the right-hand side of a context-free rule $S \rightarrow f$, in which each input slot '_' of f is replaced by a unique nonterminal N, which yields a production rule $N \rightarrow g$ for each function g that can be applied on the input slot. Nonterminals that produce the same set of right-hand sides are merged.

Example: The English function _ty-_:$\{6,7,9\} \times \{1,\ldots,9\}$ is modeled using 13 context-free rules $S \rightarrow A$ty-B, $A \rightarrow$ six, seven, nine and $B \rightarrow$ one, ..., nine. The function twenty-_:$\{1,\ldots,9\}$ is modeled as $S \rightarrow$ twenty-C with $C \rightarrow$ one, ..., nine. Since B and C produce the same words, they merge, so all rules of C are removed and twenty-C is renamed to twenty-B.

Using the described conversion, the numeral-decomposer-induced CFGs do not overgenerate. Therefore, we set GITTA's parameter $relative\_similarity\_threshold$—that controls which nonterminals can merge—to 1 to also prevent GITTA from overgeneralizing.

Fig. 5 shows the number of context-free rules induced for numerals till 99 by GITTA and the two numeral decomposer versions. Both decomposer versions outperform GITTA on average and on median, as we show in Fig. 6 and in the following table.

#(Context-free rules)	GITTA	Alg. 2	Alg. 1
Average	65.95	46.20	47.77
Median	61	44	44

In addition, GITTA does not deliver arithmetical attributes to the rules.

However, in some languages, the numeral decomposer does significantly worse than GITTA. This is because GITTA is not asked for arithmetic attributes, so it uses generalizations that the numeral decomposer considers too risky as they might cause generalizations that cannot be covered by affine linear equations. E.g., in Bavarian, the decomposers induce 101 rules, while GITTA only induces 49. Bavarian 2-digit numerals usually have masked factors and summands, so the numeral decomposer cannot unpack them. However, GITTA can still generalize $N(fa * mu)$ then, whereas the unpacking criteria forbid the numeral decomposers to do the same as $fa * mu > (fa * mu + su)/2$. GITTA also profits from generalizing empty strings, which the numeral decomposer does not dare.

6.5 Overall Statistics of Decomposition Errors

In this subsection, we attempt to determine the decomposition error rate of the numeral decomposer. A decomposition error rate is not to be confused with the word error rate of the induced grammars. While the word error rate only covers word errors, a decomposition error rate shall cover both, undergeneralization errors (Objective 1, compactness) that harm lexicon efficiency, and overgeneralization errors (Objective 2, correctness) that cause word errors.

The most straightforward measure to quantify an error rate regarding Objective 2 (correctness) is the relative frequency of numeral word with wrong number values in the induced grammars. This corresponds to the word error rate. Across all 257 languages in our dataset, the this rate is 0.775 %. However, the word error rate of our induced grammars is not very expressive regarding decomposer performance. As mentioned in Subsection 5.2, in Choapan-Zapotec, a template '_ galo' is induced that maps $x \in \{2, 3, \dots\}$ to $20 * x$, but it maps 1 to 19. If the affine linear functional equation is deduced from the value pairs $(1, 19)$ and $(2, 40)$, then it is $x \mapsto 21 * x - 2$. In this case, all numerals generated by '_ galo' get wrong number values, except for $N(1) \cdot$ 'galo' and $N(2) \cdot$ 'galo'. On the other hand, if the equation $x \mapsto 20 * x$ is deduced from the pairs $(2, 40)$ and $(3, 60)$, then only one error occurs for the numeral $N(1) \cdot$ 'galo'.

A consistent error statistic regarding Objective 2 (correctness) is the rate of templates with wrong functional equations among all templates. Across all languages, this rate is 0.325 %.

An error rate for Objective 1 (compactness) is hard to determine, as it requires understanding the numeral systems of 257 languages in order to assess which templates could possibly be covered by others.

A heuristic approach is to count how many words have not been generalized, i.e., words that have an exclusive template. Across all languages, 2.848 % of numeral words belong to an exclusive template. However, the atomic digit words—which are usually $N(1) - N(9)$—as well as exceptions obviously require their own template.

The number of unnecessary templates would be an accurate measure, but it is hard to determine for abovementioned reasons. However, we may estimate it. Recall that the induced English grammar for numerals till 999 presented at the start of Section 6 has 30 templates and appears morphologically plausible. The English numeral system has many irregularities for 2-digit numerals, but it becomes very regular for 3-digit numerals and higher. Therefore, we may assume that it has a typical number of exceptions. Considering that a morphologically plausible grammar for numerals till 999 in a language with average complexity has 30 templates, we may consider 30 as the expected number of templates needed to cover numerals till 999 in an arbitrary language.

Grammars have been induced in 257 languages. For 168 languages, the grammars cover numerals till 999, for 34 languages they cover more numerals, and for 55 languages less. Therefore, the number of templates needed to morphologically plausibly cover the numerals from all 257 languages may be estimated as $257 * 30 = 7710$. As all the induced numeral grammars actually have 9854 templates combined, we may estimate that $(9854 - 7710)/(9854) = 21.758\,\%$ of the templates are unnecessary. We acknowledge that this percentage can easily fluctuate by 10 percentage points if we misesitamate the complexity of the English numeral system by even $10\,\%$.

Overall, $0.325\,\%$ of templates are overgeneralizing and about $21.8\,\%$ of templates are undergeneralizing. They cause word errors and lexicon inefficiency, respectively. Combined, we yield a per-template decomposition error rate—not to be confused with the word error rate—of $22\,\%$. Note, that this number is not a classical per-input error rate, as it does not give the per-input rate of inputs (words) that lead to a wrong output (template) but the per-output rate of erroneous outputs. We expect the rate per input word to be lower because most of the erroneous output templates are undergeneralizing. This implies that most erroneous templates account for a lower number of words than the correct templates.

7. SUMMARY

We showed that an arithmetic-based numeral decomposer can work universally across language and outperform more general state-of-the-art approaches in numeral grammar induction. We have justified criteria with respect to Hurford's Packing Strategy to detect the factor and the summand word of a numeral word. Given S is a subnumeral of N, we found that

if $s \leq \sqrt{n}$, then S must be (part of) N's factor or summand word,

if $\sqrt{n} < s < n/2$, then S could be (part of) N's summand word and

if $n/2 < s$, then S cannot be part of N's factor or summand word.

The criteria have been applied in two decomposition algorithms[8] that were tested for incremental grammar induction in 257 languages which are listed in Appendix A.

The advanced numeral decomposer induces plausible numeral grammars in a great variety of natural languages. In 2 out of 3 cases, its induced CFGs are more compact than CFGs induced by the state-of-the-art grammar induction algorithm GITTA (Winters & Raedt, 2020). The main limitation of the numeral-decomposer induced grammars is that they only allow for generalization of entire subnumerals. In languages like Kartvelian, Hayastani, or Bavarian, numerals often drop or change letters when used as subnumerals, so they cannot be detected and generalized, which significantly enlarges the numeral-decomposer induced grammars.

In Bulgarian, Birom, and Yoruba, we compared numeral-decomposer induced grammars to expert made gold standard grammars (Derzhanski & Veneva, 2020). All three induced grammars are similar or equal to the gold standards. Specifically, they yield higher accuracies than the grammars that Hammarström's k-cluster algorithm had deduced.

[8] The source code of both algorithm versions is published in I. K. Maier (2023).

The numeral-decomposer induced grammars have the inherent advantage over general syntactic grammar induction algorithms of parallelly induced arithmetical attributes. In 243 out of 257 languages, the induced arithmetical attributes were entirely correct. Incorrect arithmetic was mainly induced in such numerals in which a misunderstanding is also conceivable for humans.

Another advantage of the numeral decomposer is that it can decompose numerals incrementally in a learning process. Syntactic grammar induction requires comparisons of expressions, like 'twenty-one', with other expressions of the same abstraction level, like 'twenty-two', to find patterns for generalization. In contrast, the numeral decomposer just needs to know the expressions of the lower abstraction level, e.g., when it knows that 'one' is the numeral of 1, then it understands that 'one' is a generalizable part of 'twenty-one' based on the unpacking criteria. Incrementality facilitates the expansion of existing grammars. Most existing grammar induction methods are nonincremental (Muralidaran et al., 2021).

8. OUTLOOK

The presented numeral decomposition algorithm produced correct numeral grammars in 243 languages and it can be used for any language. The numeral grammars can serve as valuable assets for low-resource languages, as they can be integrated into NLP pipelines to enhance named entity recognition, which supports data-driven language models. Further extensions of this work can support NLP even more.

Two major limitations are that the tests have only been conducted on grammatical standard forms of numerals and that the numeral decomposer cannot unpack and generalize subnumerals that appear in a masked or inflected form. Both shortcomings could be dealt with by letting the numeral decomposer learn several grammatical forms of each numeral. In this way, a stem of all forms could be determined. In the process, the numeral decomposer could detect and evaluate not only fully contained subnumerals, but also just stems of such subnumerals. Depending on the degree of tolerance, it may therefore detect and unpack the 'thir' in 'thirteen' if it has learned before that 'third' is a grammatical variant of $N(3)$.

Such measures could greatly support generalizations. It could even prevent overgeneralizations in cases where the tolerance helps detect $N(fa * mu)$. E.g., when 'quatre-vingt**s**' is detected in 'quatre-vingt-deux', the problem dealt with in Subsection 5.3.2—which partly persists—does not come up. On the other hand, it can cause that substrings are unintentionally detected as subnumerals due to a random similarity, which may cause deeper problems.

The numeral decomposer may be tested on learning numerals in a random unchronological order. The test poses the challenge of working with a limited lexicon. When the decomposer gets to learn 'sixteen' before 'six', it cannot detect and unpack 'six', which leads to a shortcoming in generalization. The shortcoming could be quantified by learning numerals in a random order that uses a suitable probability distribution. Unchronological learning could also be dealt with by giving up incrementality and decomposing 'sixteen' again after 'six' got learned.

The numeral-decomposer-based incremental learning algorithm could also involve reinforcement. For a learned template like '_ty-_' the learning algorithm could think up words by inserting alternative subnumerals into the slots. It just needs some sort of supervisor that accepts or rejects generalizations of learned words. Such a reinforcement learning offers possibilities for applicationrelated projects:

- If the supervisor is replaced by a human, the reinforcement learning algorithm can work like a chatbot that can create generative grammars for number words in low-resource languages with human support. The human would only need to answer questions like 'What is the numeral of number x?' and 'Does numeral X exist?'.

- If the learner is able to extract numerals out of text data, only answers to questions of the form 'Does numeral X exist?' would be needed. And these questions could be answered with a search engine and a statistical model, which—given a numeral X and the number of search results for X—could decide if X is a correctly spelled numeral.

Given that our error report (Subsection 5.2) names context-sensitive numerals, their authenticity and potential implications may be discussed further by linguists.

ACKNOWLEDGEMENTS

We would like to thank Professor Günther Wirsching from the Catholic University of Eichstätt, and our colleagues Martin Behm and Johannes Kuhn, for their valuable advice and contributions.

REFERENCES

Akinadé, O. O., & Ọdẹ́jọbí, Ọ. A. (2014). Computational modelling of yorùbá numerals in a number to-text conversion system. Journal of Language Modelling, 2(1), 167–211.

Alkhazi, I. S. B. (2019). Compression-based parts-of-speech tagger for the arabic language. International Journal of Computational Linguistics (IJCL), 10, 1–15. https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJCL-95

Andersen, H. (2004). The plasticity of universal grammar. Convergence. Interdisciplinary Communications,2005, 216.

Anderson, C. (2019). Numerical approximation using some. Proceedings of Sinn und Bedeutung, 19, 54–70. https://ojs.ub.uni-konstanz.de/sub/index.php/sub/article/view/221

Brainerd, B. (1966). Grammars for number names. Foundations of Language, 2(2), 109–133. Retrieved March 14, 2025, from http://www.jstor.org/stable/25000213

Carroll, G., & Charniak, E. (1992). Two experiments on learning probabilistic dependency grammars from corpora. Department of Computer Science, Univ. https : / / doi . org / 10 . 5555 /864689

Chorozoglou, Z., G., N. Z., E. C., Papakitsos, Galiotou, E., & Giovanis, A. (2021). Review of parsing in modern greek - a new approach. International Journal of Computational Linguistics (IJCL), 12, 1–8. https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJCL-119

Dékány, É. (2025). Anatomy of a complex numeral: Overcounting, with special attention to ch'ol. https://doi.org/https://doi.org/10.7280/S9RV0KRH

Derzhanski, I. (2025). Fifty-eight. Proceedings of the Annual International Conference of the Institute for Bulgarian Language. https://doi.org/10.7546/ConfIBL2025.16

Derzhanski, I., & Veneva, M. (2018). Linguistic problems on number names. Proceedings of the Third International Conference on Computational Linguistics in Bulgaria (CLIB 2018), 169–176. https://www.researchgate.net/publication/324362714_Linguistic_Problems_on_Number_Names

Derzhanski, I., & Veneva, M. (2020). Generating natural language numerals with TeX. Proceedings of the Fourth International Conference on Computational Linguistics in Bulgaria (CLIB2020), 112–120. https://aclanthology.org/2020.clib-1.12/

Drozdov, A., Verga, P., Yadav, M., Iyyer, M., & McCallum, A. (2019). Unsupervised latent tree induction with deep inside-outside recursive autoencoders [ArXiv]. https://arxiv.org/abs/1904.02142

Dryer, M. S., & Haspelmath, M. (Eds.). (2013). Wals online (v2020.4). Zenodo. https://doi.org/10.5281/zenodo.13950591

Flach, G., Holzapfel, M., Just, C., Wachtler, A., & Wolff, M. (2000). Automatic learning of numeral grammars for multi-lingual speech synthesizers. 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100), 3, 1291–1294. https://doi.org/10.1109/ICASSP.2000.861814

Friedman, D., Wettig, A., & Chen, D. (2022). Finding dataset shortcuts with grammar induction. Empirical Methods in Natural Language Processing (EMNLP). https://doi.org/10.18653/v1/2022.emnlp-main.293

Gil, D. (2013a). Distributive numerals (v2020.4). In M. S. Dryer & M. Haspelmath (Eds.), The world atlas of language structures online. Zenodo. https://doi.org/10.5281/zenodo.13950591

Gil, D. (2013b). Numeral classifiers (v2020.4). In M. S. Dryer & M. Haspelmath (Eds.), The world atlas of language structures online. Zenodo. https://doi.org/10.5281/zenodo.13950591

Graben, P. b., Römer, R., Meyer, W., Huber, M., & Wolff, M. (2019). Reinforcement learning of minimalist numeral grammars. 2019 10th IEEE International Conference on Cognitive Info communications (CogInfoCom), 67–72. https://doi.org/10.1109/CogInfoCom47531.2019.9089924

Hammarström, H. (2008). Deduction of Numeral Grammars. https://www.academia.edu/3142323/ Deduction_of_Numeral_Grammars

Htut, P. M., Cho, K., & Bowman, S. R. (2018). Grammar induction with neural language models: An unusual replication [ArXiv]. https://arxiv.org/abs/1808.10000

Hurford, J. (2007). A performed practice explains a linguistic universal: Counting gives the packing strategy. Lingua, 117, 773–783. https://doi.org/10.1016/j.lingua.2006.03.002

Hurford, J. (2011). The linguistic theory of numerals (Vol. 16). Cambridge University Press. https://doi.org/10.1017/S0022226700005776

Ionin, T., & Matushansky, O. (2006). The Composition of Complex Cardinals. Journal of Semantics, 23(4), 315–360. https://doi.org/10.1093/jos/ffl006

Ivani, J. K. (2017). The morpho syntax of number systems: A cross-linguistic study [Doctoral dissertation, Università degli studi di Bergamo].

Jon-And, A., & Michaud, J. (2024). Usage-based grammar induction from minimal cognitive principles. Computational Linguistics, 50(4), 1375–1414. https://doi.org/10.1162/coli_a_00528

Khamdamov, U., Mukhiddinov, M., Akmuradov, B., & Zarmasov, E. (2020). A novel algorithm of numbers to text conversion for uzbek language tts synthesizer. 2020 International Conference on Information Science and Communications Technologies (ICISCT), 1–5. https://doi.org/10.1109/ICISCT50599.2020.9351434

Kim, Y., Dyer, C., & Rush, A. (2019, July). Compound probabilistic context-free grammars for grammar induction. In A. Korhonen, D. Traum, & L. Màrquez (Eds.), Proceedings of the57th annual meeting of the association for computational linguistics (pp. 2369–2385). Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1228

Klein, D., & Manning, C. D. (2001). Natural language grammar induction using a constituent context model. In T. Dietterich, S. Becker, & Z. Ghahramani (Eds.), Advances in neural information processing systems (Vol. 14). MIT Press. https : / / proceedings . neurips cc /paper_files/paper/2001/file/2d00f43f07911355d4151f13925ff292-Paper.pdf

Li, B., Corona, R., Mangalam, K., Chen, C., Flaherty, D., Belongie, S., Weinberger, K., Malik,J., Darrell, T., & Klein, D. (2024, June). Re-evaluating the need for visual signals in unsupervised grammar induction. In K. Duh, H. Gomez, & S. Bethard (Eds.), Findings of the association for computational linguistics: Naacl 2024 (pp. 1113–1123). Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.findings-naacl.70

Maier, I., & Wolff, M. (2022). Poster: Decomposing numerals. EUNICE Science Dissemination: Poster Competition. https://doi.org/10.5281/zenodo.7501698

Maier, I. K. (2023). Numeral Decomposer 1.1 [GitHub]. https://github.com/ikmMaierBTUCS /Numeral-Decomposer-1.1/

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge University Press.

Martí, L. (2020). Numerals and the theory of number. Semantics and Pragmatics, 13(3), 1–57.https://doi.org/10.3765/sp.13.3

Mendia, J. A. (2018). Epistemic numbers. Proceedings of SALT, 28. https://doi.org/10.3765/salt.v28i0.4433

Muralidaran, V., Spasic, I., & Knight, D. (2021). A systematic review of unsupervised approaches to grammar induction. Natural Language Engineering, 27, 647–689. https://doi.org/10.1017/S1351324920000327

Rhoda, İ. A. (2017). Computational analysis of igbo numerals in a number-to-text conversion system. Journal of Computer and Education Research, 5(10), 241–254. https://doi.org/10.18009/jcer.325804

Seginer, Y. (2007, June). Fast unsupervised incremental parsing. In A. Zaenen & A. van den Bosch(Eds.), Proceedings of the 45th annual meeting of the association of computational linguistics(pp. 384–391). Association for Computational Linguistics. https://aclanthology.org/P07-1049/

Shen, Y., Tan, S., Sordoni, A., & Courville, A. (2019). Ordered neurons: Integrating tree structures into recurrent neural networks [ArXiv]. https://doi.org/10.48550/arXiv.1810.09536

Sproat, R. (2022). Boring Problems Are Sometimes the Most Interesting. Computational Linguistics,48(2), 483–490. https://doi.org/10.1162/coli_a_00439

Stolcke, A., & Omohundro, S. M. (1994). Inducing probabilistic grammars by bayesian model merging[ArXiv]. https://api.semanticscholar.org/CorpusID:7324510

Stolz, T., & Veselinova, L. N. (2013). Ordinal numerals (v2020.4). In M. S. Dryer & M. Haspelmath(Eds.), The world atlas of language structures online. Zenodo. https://doi.org/10.5281/zenodo.13950591

Sumamo, J. S., & Teferra, S. (2018). Designing a rule based stemming algorithm for kambaata language text. International Journal of Computational Linguistics (IJCL), 9, 41–54. https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJCL-93

Veselinova, L. N. (2020). Numerals in morphology. Oxford Research Encyclopedia of Linguistics.https://doi.org/10.1093/acrefore/9780199384655.013.559

Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. Journal of the ACM,21(1), 168–173. https://doi.org/10.1145/321796.321811

Winters, T., & Raedt, L. D. (2020). Discovering textual structures: Generative grammar induction using template trees [ArXiv]. https://arxiv.org/abs/2009.04530

Zabbal, Y. (2005). The syntax of numeral expressions. Ms., University of Massachusetts, Amherst. https://api.semanticscholar.org/CorpusID:214631884

Zhao, Y., Fei, H., Wu, S., Zhang, M., Zhang, M., & Chua, T.-s. (2025). Grammar induction from visual, speech and text [ArXiv]. https://arxiv.org/abs/2410.03739

Žoha, L., Wągiel, M., & Caha, P. (2022). The morphology of complex numerals: A cross-linguistic study. LingBaW. Linguistics Beyond and Within, 8, 200–217.

A. LIST OF DATA SETS (LANGUAGES)

The languages of the data sets are written in parentheses when they are not obvious.

Acholi	Adyghe	Afrikaans	Albanian (Shqiperian)
Aloapam-Zapotec	Alsatian	Alutiiq	Amharic
Antillean-Creole-Of-Martinique	Arabic	ar (Arabic)	Araki
Arberesh	Arhuaco	Arikara	Armenian (Hayastani)
Assiniboine	Asturian	Aukan	Awa-Pit
Aymara	Azerbaijani	Baka	Bambara
Bashkir	Basque	Bavarian	Belarusian
Bezhta	Birom	Breton	Bulgarian
Burushaski	Calo	Cape-Verdean-Creole	Carrier (Dakelh)
Catalan	Central-Tarahumara	Chavacano	Cherokee
Choapan-Zapotec	Chol	Chuvash	Cocama
Comox	Copala-Triqui	Cornish	Corsican
Crimean-Tatar	Czech	cs (Czech)	Dagbani
Danish	da (Danish)	Dogrib (Tłı̨chǫ)	Dzambazi-Romani
English	en_GB (British English)	en_IN (Indian English)	Eonavian
Estonian	Faroese	Finnish (Suomi)	fi (Suomi)
French	fr (French)	fr_BE (Belgian French)	fr_CH (Swiss French)
fr_DZ (Algerian French)	Friulian	Ga	Galician
Gallo	Garifuna	Georgian (Kartvelian)	German (Deutsch)
de (Deutsch)	Gilbertese (Kiribati)	Gottscheerish	Guarani
Gwere	Haida	Haitian-Creole	Halkomelem
Hausa	he (Hebrew)	Hopi	Hungarian (Magyar)
Hunsrik	Hupa	Icelandic	Igbo
Inari-Sami	Indonesian	id (Indonesian)	Ingrian
Ingush	Innu	Inupiaq	Irish
Isthmus-Zapotec	Italian	it (Italian)	Jakaltek
Japanese (Nihongo)	ja (Nihongo)	Jaqaru	Jerriais
Kabiye	Kalderash-Romani	Kalina	kn (Kannada)
Kaqchikel	Karelian	Kazakh	Kiliwa
Kirmanjki	Kituba	Klallam	Koasati
ko (Korean)	Kristang	Kutenai	Kven
Kyrgyz	Lachixio-Zapotec	Ladin	Lakota
Lango	Latin	Latvian	lv (Latvian)
Laz	Lezgian	Lingala	Lithuanian
lt (Lithuanian)	Livonian	Llanito	Lombard-Milanese
Lower-Sorbian	Lowland-Oaxaca-Chontal	Lule-Sami	Lushootseed
Luxembourgish	Macedonian	Makhuwa	Maltese
Mandinka	Manx-Gaelic	Maori	Mapudungun
Marshallese	Mauritian-Creole	Mazahua	Menominee
Miami-Illinois	Michif	Micmac	Minangkabau
Mohawk	Mohegan-Pequot	Moloko	Mussau-Emira
Mwani	Navajo	Ndom	Nelemwa
Nengone	Nigerian-Fulfulde	nl (Nederlands)	North-Frisian
Northern-Kurdish	Northern-Sami	Northern-Yi	Norwegian-Bokmal
no (Norwegian)	Nume	Nyungwe	Occitan
Ojibwa	Okanagan	Oneida	Oromo
Paici	Pennsylvania-German	Persian (Farsi)	Picard
Pite-Sami	Plautdietsch	Polari	Polish
pl (Polish)	Portuguese-Brazil	pt_BR (Brazilian Portuguese)	Portuguese-Portugal
pt (Portuguese)	Proto-Indo-European	Punu	Purepecha
Quetzaltepec-Mixe	Rapa-Nui	Rincon-Zapotec	Romani
ro (Romanian)	Romansh	Russian	ru (Russian)
Saanich	Sango	Santa-Ana-Yareni-Zapotec	Sardinian
Saterland-Frisian	Scots	Scottish-Gaelic	Serbian
sr (Serbian)	Shona	Shuswap	Sierra-Otomi
Siletz-Dee-Ni	Skolt-Sami	Slovak	Slovene
sl (Slovene)	Soga	Somali	Soninke
South-Efate	Southern-Quechua	Southern-Sami	Spanish
es (Spanish)	es_CO (Columbian Spanish)	es_VE (Venezuelan Spanish)	Squamish
Sranan-Tongo	Susu	Swahili	Swedish
Swiss-German	Tahitian	Tamazight	Tetun-Dili
Tezoatlan-Mixtec	th (Thai)	Timbisha	Tlingit
Tok-Pisin	Tolowa	Tongan-Telephone-Style	Totontepec-Mixe
Tsez	Tsonga	Tswana	Tukudede
Tunica	Turkish	tr (Turkish)	Ukrainian
uk (Ukrainian)	Ume-Sami	Upper-Sorbian	Uyghur
Venetian	Veps	vi (Vietnamese)	Votic
Wayuu	Welsh	West-Frisian	Wymysorys
Xhosa	Yakut (Sakha)	Yao	Yiddish
Yoruba	Yupik	Zulu	

B. YORUBA AND BULGARIAN INDUCED AND EXPERT-MADE GRAMMARS

Bulgarian induced grammar

Rule/Function	Values
ATOMS	$1,\ldots,12,20$
_nadeset $(x) \mapsto x+10$	$13,\ldots,19$
dvadeset i _ $(x) \mapsto x+20$	$21,\ldots,29$
_deset $(x) \mapsto 10x$	$10x$ for $x \in \{3,\ldots,9\}$
_deset i _ $(x,y) \mapsto 10x+y$	$10x+y$ for $x \in \{3,\ldots,9\}$, $y \in \{1,\ldots,9\}$

Expert-made grammar

Rule/Function	Values
ATOMS	$1,\ldots,12,20$
_nadeset $(x) \mapsto x+10$	$13,\ldots,19$
dvadeset i _ $(x) \mapsto x+20$	$21,\ldots,29$
_deset $(x) \mapsto 10x$	$10x$ for $x \in \{3,\ldots,9\}$
_deset i _ $(x,y) \mapsto 10x+y$	$10x+y$ for $x \in \{3,\ldots,9\}$, $y \in \{1,\ldots,9\}$

Yoruba induced grammar

Rule/Function	Values
ATOMS	$1,\ldots,10,20$
ogun _ $(x) \mapsto 20x$	$20x$ for $x \in \{2,\ldots,9\}$
_ l-e.wa $(x) \mapsto x+10$	$11,\ldots,14$
_ l-ogun $(x) \mapsto x+20$	$21,\ldots,24$
_ l-ogun eji $(x) \mapsto x+40$	$41,\ldots,44$
_ dinogun _ $(x,y) \mapsto x+10y$	$30,\ldots,34$
_ din_ _ (x,y,z) $\mapsto x-y+20z$	$x-20+20z$ for $x \in \{10,\ldots,14\}$ $z \in \{3,\ldots,10\}$
_ din ogun $(x) \mapsto -1x+20$	$15,\ldots,19$
_ din e.wa dinogun _ $(x,y) \mapsto 25$	25
_ din _ dinogun _ $(x,y,z) \mapsto -x+3y$	$26,\ldots,29$
_ din _ din_ _ (x,y,z,a) $\mapsto -x-y+20a$	$-x-10+20a$ for $x \in \{1,\ldots,5\}$ $a \in \{3,\ldots,10\}$
_ _ $(x,y) \mapsto x+20y-20$	$x+20y-20$ for $x \in \{15,\ldots,24\} \setminus \{20\}$ $y \in \{3,\ldots,10\}$

Expert-made grammar

Rule/Function	Values
ATOMS	$1,\ldots,10,20$
ogun _ $(x) \mapsto 20x$	$20x$ for $x \in \{2,\ldots,9\}$
_ l-_ $(x,y) \mapsto x+y$	$x+y$ for $x \in \{10,20,\ldots,180\}$ $y \in \{1,\ldots,4\}$
e.wa din ogun _ $(x) \mapsto 20x-10$	$20x-10$ for $x \in \{2,\ldots,9\}$
_ din _ $(x,y) \mapsto x-y$	$x-y$ for $x \in \{20,\ldots,180\}$ $y \in \{1,\ldots,5\}$