Generation and Parsing of Number to Words in Tamil

Authors: Muthiah Annamalai, Sathia Mahadevan

Abstract

We discuss generation and parsing of Tamil words from and to numbers respectively. We show this task can be achieved in O(n) complexity and performs well for written text. We explore features of these algorithms within a voice user-interface based calculator; we also list some of the limitations. These algorithms are implemented in the Open-Tamil library.

1. Introduction

Automatic evaluation of mathematical expressions in the context of natural-language processing (NLP) has been carried out in English language as far back as the 1960s [1]. Voice interfaces to various banking, token based queue systems in retail and wholesale spaces, are desirable to read-out numbers as words in the specific language; conversely systems interacting by speech input are also desired to take number input as spoken words in the specific language. To this effect we need systems to generate the numbers to words and parse text back to numbers.

Not much work is reported on number to word generation in Indian languages in general and Tamil in particular, to best of our knowledge. Tamil numbers have specific descriptions in written and spoken forms of language [2], [6], [7].

2. Generation

Number forms in Tamil have recursive properties and details of their nuances in dealing with 90s, 900s, 9000s numbers and 10-19 numbers are elaborated in standard reference books like [2], [6], [7].

Typically, multiplication tables (வாய்பாடு) are used in grade school (K-12) to commit basic mathematical facts to memory and take a textual form as “இரண்டு நான்கு எட்டு", meaning 2 x 4 = 8. Ability to generate numbers in word-forms enables us to generate these multiplication tables ad-hoc. [4], [5].

Fig. 1. Multiplication table for 6 created using this numeral generation algorithm in [4].

2.1. Algorithm

Algorithm works for generating integral and floating point non-negative numbers; it generates text forms of numbers in both Indian (i.e. using *lakhs*, *crores*) and American standard (i.e. using *millions*, *billions*). Lakh is 100,000, Crore is 100 Lakhs, and Million is 10 Lakhs. We present the algorithm for Indian standard number to word conversion using base-10:

Input: floating point number ***N***
Output: string of Tamil words ***T***

1. Load list of prefix and string suffix for all Tamil number words - 63 words in all.
2. Find the quotient **Q**, remainder **R** for ***N*** divided by 1 crore, lakh, thousand, hundreds, or tens
3. If **Q** is zero set ***N*=R** and continue to 2.
4. Convert the quotient to words ***T***
    a. Take special care to handle 90s, 900s, 9000s, correctly.
    b. Take special care to handle number in 11-19.
5. Invoke same algorithm recursively for remainder **R**.
6. Concatenate results from 5 to ***T***
7. Return **T**

Similarly we get the American standard number to word conversion if we replace step 2 in above algorithm to divide by trillion, billion and million in replacement of crore, lakh placeholder values.

The code for this algorithm is shown in subroutine **num2tamilstr** in Appendix **A.1**. The code for American standard number to word generation code is similar but not shown in this paper; one can refer to [4b].

2.2. Performance

The current algorithm is O(n) for *n* significant digits of the input specified as floating-point number. The decimal point, pulli, is handled correctly and the current algorithm generates a much larger verbosity of full-precision floating point input upto the number of significant digits present in the computer memory representation of double.

3. Parsing

3.1 Algorithms

Algorithm works for parsing integral and floating point non-negative numbers; it parses text forms of numbers in both Indian (i.e. using *lakhs*, *crores*) and American standard (i.e. using *millions*, *billions*). It is roughly the converse of the algorithm in section 2.

Input: string of Tamil list of words ***T***
Output: floating point number ***N***

1. Load list of prefix and string suffix for all Tamil number words - 63 words in all.
2. Initialize N at 0
3. Create temporary stack **S**
4. FOR word W in **T**
5. IF W in stop words (crores, lakhs, thousands, hundreds, tens)
    a. Convert words in stack **S** into value and scale temporary result ***N*** using a helper routine which handles input upto value 100,0000.
    b. Empty stack **S**
6. ELSE: push **W** into **S**
7. END loop started at 4.
8. Stack S is mostly non-empty and you have to use a helper routine to get the final portion of the number using the same helper function in 5a.
9. Correctly parsed value is stored in ***N***

Helper routine also understands decimal point (*pulli*) and returns a single number which forms a scale for the intermediate multiple of 10. The code for the algorithm and helper algorithm are shown in subroutines **tamilstr2num** and **helper_tamilstr2num** respectively in Appendix. **A**.

3.2 Performance

This algorithm is linear in complexity O(n) for the number of words of text in the input. The accuracy of the parsed number depends on the floating point representation of the hardware - and code uses 64-bit double representation as implemented now.

4. Results and Applications

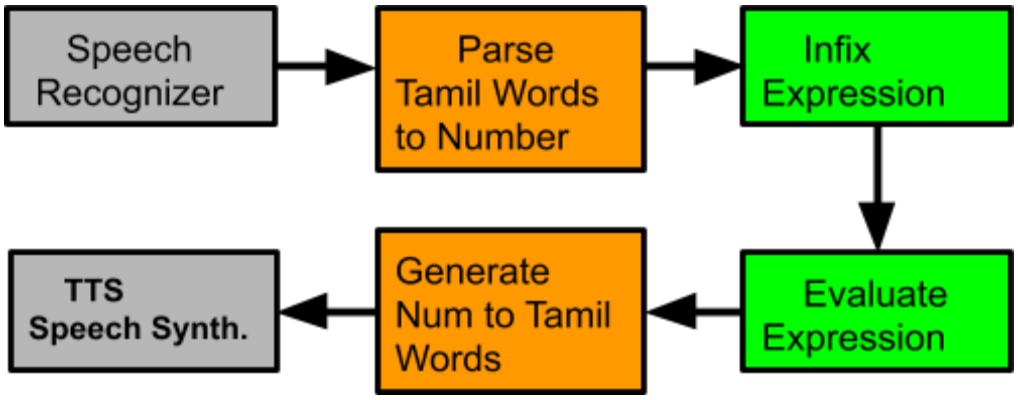

Fig. 2. Voice User-Interface for Tamil Spoken Calculator.

The concept of a spoken voice user-interface (VUI) calculator is shown in Fig. 2. While such a system has been proposed for English and other languages, e.g. in patent applications [3a,b] etc. our work forms the foundation of such a system in Tamil. Key limitations of [3] are the requirement of a generator/parser for numerals in the target language as well as the speech recognition and synthesis facilities.

Using or extending this technique to associating numerals, ordinals, and number conjugates [7] to nouns in a corpus will help improve the resolution and fine-tune named entity recognizers, by adding these quantitative attributes on such named-entities.

Table. 1. Expression Evaluation

Input Expression	Parsed Expression	Evaluated Result	Output Result
ஒன்று கூட்டல் ஒன்று	1 + 1	2	இரண்டு
ஒன்று கூட்டல் இரண்டு பெருக்கல் பத்து	1 + 2 * 10	21	இருபத்தொன்று
ஓர் ஆயிரம் கழித்தல் ஐந்து பெருக்கல் (ஒன்பது கூட்டல் ஒன்று)	1000 - 5*(9 + 1)	950	தொள்ளாயிரத்து ஐம்பது
ஒரு இலட்சம் பெருக்கல் பத்து	100000*10	1000000	பத்து இலட்சம்

4.1. Results

To this extent we show core concepts (orange, green coded blocks) of such spoken mathematical expressions in Tamil and their evaluation with Open-Tamil library application named *olini (ஒலினி)* [4]. This expression evaluator understands infix expressions including binary operators of addition, subtraction, multiplication and division including parentheses. While a full integration of the speech recognizer (ASR) and text-to-speech (TTS) modules have not been accomplished, the resulting performance can be summarized in Table. 1.

4. 2. Limitations

Current work is limited to perfectly spelled text form of numbers, and does not work for text input in spoken Tamil dialect, or text with spelling errors. However, there are simple remedies to these contexts.

Future work, could include parsing of text with morphological analyzer and tolerance toward spelling errors in text by using spelling corrector. Further work, can also involve moving towards generation/recognition of the various forms of Tamil number words - fractions (அரை,கால், முக்கால், ... மற்றும் பல), ordinals, multiplication tables (வாய்பாடு), domain specific uses in colloquial spoken contexts e.g. “ஒன்றையும் பத்தையும் கூட்டு", "பத்தில் நாலை கழி," [2],[7] etc.

We hypothesize the algorithm for generating number to words is identical in other Dravidian languages due to the underlying relationships of the languages. We suspect this may also hold true for TTS generation of audio of the Numerals. This remains to be seen.

5. Conclusion

This work presents, for the first time to our knowledge, a comprehensive account of generating and parsing Tamil number words. We show applications of our work to a future voice user-interface based spoken calculator, and even perhaps name-entity recognizers.

6. References

1. Charles J. Swift, “Evaluating numbers expressed as strings of English words,” CACM Oct. (1960).
2. Vasu Renganathan, “Tamil Language in Context,” (2011).
3. (a) Patent - US 4,882,685 “Voice Activated Electronic Calculator,” van der Lely (1989).
   (b) Patent - US 2008.0312928A1 “Natural Language Speech Recognition Calculator,” Goebel, Shivanna (2008).
4. (a) Tamilpesu.us - Open-Tamil Indic Computing Platform; http://tamilpesu.us (accessed June 5th 2020).
   (b) Open-Tamil Git Repository - https://github.com/Ezhil-Language-Foundation/open-tamil (accessed June 2020).
5. S. Abuthahir, et-al “Growth and Evolution of Open-Tamil,” Tamil Internet Conference, Coimbatore, Tamilnadu. (2018).

6. W. H. Arden, “A Progressive Grammar of Common Tamil,” Soc. for Promoting Christian Knowledge (1910).
7. சிங்கப்பூர் சித்தார்த்தன், "இலகு தமிழில் இனிக்கும் தமிழ் இலக்கணம்," நர்மதா பதிப்பகம் (2017). Siddarthan, “A treatise and guide to learn Tamil Grammar,” Narmada Publications (2017).

A. Appendix - Python Source Code

A.1 Generating Tamil Number Words

```
def num2tamilstr( *args ):
    """ work till one lakh crore - i.e 1e5*1e7 = 1e12.
        turn number into a numeral, Indian style. Fractions upto 1e-30"""
    number = args[0]
    if len(args) < 2:
        filenames = []
    else:
        filenames = args[1]
    if len(args) ==3:
        tensSpecial = args[2]
    else:
        tensSpecial='BASIC'
    if not any( filter( lambda T: isinstance( number, T), [str,int, float]) ) or
isinstance(number,complex):
        raise Exception('num2tamilstr input has to be a integer or float')
    if float(number) > int(1e12):
        raise Exception('num2tamilstr input is too large')
    if float(number) < 0:
        return u"- "+num2tamilstr( -number )

    units = (u'பூஜ்ஜியம்', u'ஒன்று', u'இரண்டு', u'மூன்று', u'நான்கு', u'ஐந்து', u'ஆறு', u'ஏழு', u'எட்டு',
u'ஒன்பது', u'பத்து') # 0-10
    units_suffix = (u'பூஜ்ஜியம்', u'தொன்று', u'திரண்டு', u'மூன்று', u'நான்கு', u'தைந்து', u'தாறு', u'தேழு',
u'தெட்டு', u'தொன்பது', u'பத்து') # 0-10
    units_suffix_nine = (u'பூஜ்ஜியம்', u'றொன்று', u'றிரண்டு', u'மூன்று', u'நான்கு', u'றைந்து', u'றாறு',
u'றேழு', u'றெட்டு', u'றொன்பது', u'பத்து') # 0-10
    tween = [1.0,2.0,5.0,6.0,7.0,8.0,9.0]
    teens = (u'பதினொன்று', u'பனிரண்டு', u'பதிமூன்று', u'பதினான்கு', u'பதினைந்து',u'பதினாறு',
u'பதினேழு', u'பதினெட்டு', u'பத்தொன்பது') # 11-19
    tens = (u'பத்து', u'இருபது', u'முப்பது', u'நாற்பது', u'ஐம்பது',u'அறுபது', u'எழுபது', u'எண்பது',
u'தொன்னூறு') # 10-90
    tens_full_prefix = (u'இருபத்து', u'முப்பத்து', u'நாற்பத்து', u'ஐம்பத்து', u'அறுபத்து', u'எழுபத்து',
u'எண்பத்து', u'தொன்னூற்று') # 10+-90+
    tens_prefix = (u'இருபத்', u'முப்பத்', u'நாற்பத்', u'ஐம்பத்', u'அறுபத்', u'எழுபத்', u'எண்பத்',
u'தொன்னூற்') # 10+-90+
    hundreds = ( u'நூறு', u'இருநூறு', u'முன்னூறு', u'நாநூறு',u'ஐநூறு', u'அறுநூறு', u'எழுநூறு',
u'எண்ணூறு', u'தொள்ளாயிரம்') #100 - 900
    hundreds_suffix = (u'நூற்றி', u'இருநூற்று', u'முன்னூற்று', u'நாநூற்று', u'ஐநூற்று', u'அறுநூற்று',
u'எழுநூற்று', u'எண்ணூற்று',u'தொள்ளாயிரத்து') #100+ - 900+

    one_thousand_prefix = u'ஓர்'
```

```
thousands = (u'ஆயிரம்',u'ஆயிரத்து')

one_prefix = u'ஒரு'
lakh = (u'இலட்சம்',u'இலட்சத்து')
crore = (u'கோடி',u'கோடியே')

pulli = u'புள்ளி'
n_one = 1.0
n_ten = 10.0
n_hundred = 100.0
n_thousand = 1000.0
n_lakh = 100.0*n_thousand
n_crore = (100.0*n_lakh)
# handle fractional parts
if float(number) > 0.0 and float(number) < 1.0:
    rval = []
    rval.append(pulli)
    filenames.append( 'pulli' )
    number_str = str(number).replace('0.','')
    for digit in number_str:
        filenames.append( "units_%d"%int(digit))
        rval.append( units[int(digit)] )
    return u' '.join(rval)

if isinstance(number,str):
    result = u""
    number = number.strip()
    assert(len(args) == 1)
    assert(len(number) > 0)
    is_negative = number[0] == "-"
    if  is_negative:
        number = number[1:]
    frac_part = u""
    if number.find(".") >= 0:
        rat_part,frac_part = number.split(".")
        frac_part = num2tamilstr(u"0."+frac_part)
    else:
        rat_part = number
    if len(rat_part) > 0:
        result = num2tamilstr(float(rat_part))
    result = result +u" "+ frac_part
    return is_negative and "-" + result.strip() or result.strip()

suffix_base = { n_crore: crore, n_lakh : lakh, n_thousand : thousands}
suffix_file_map = {n_crore: "crore", n_lakh : "lakh", n_thousand : "thousands"}

file_map = {n_crore :["one_prefix","crore_0"],
```

```
            n_lakh : ["one_prefix","lakh_0"],
            n_thousand :  ["one_thousand_prefix", "thousands_0"],
           n_hundred : ["hundreds_0"], #special
           n_ten : ["units_10"],
           n_one : ["units_1"]}

num_map = {n_crore : [one_prefix,crore[0]],
           n_lakh  : [one_prefix,lakh[0]],
           n_thousand : [one_thousand_prefix, thousands[0]],
           n_hundred : [hundreds[0]], #special
           n_ten : [units[10]],
           n_one : [units[1]]}

all_bases = [n_crore, n_lakh, n_thousand, n_hundred, n_ten,n_one]
allowed_bases = list(filter( lambda base: number >= base, all_bases ))
if len(allowed_bases) >= 1:
    n_base = allowed_bases[0]
    if number == n_base:
        if tensSpecial=='BASIC':
            filenames.extend(file_map[n_base])
            return u" ".join(num_map[n_base])
        elif tensSpecial=='NINES':
            filenames.extend(file_map[n_base])
            return units_suffix_nine[int(number%10)]
        else:
            filenames.extend(file_map[n_base])
            return units_suffix[int(number%10)]
    quotient_number = int( number/n_base )
    residue_number = number - n_base*quotient_number
    #print number, n_base, quotient_number, residue_number, tensSpecial
    if n_base == n_one:
        if isinstance(number,float):
            int_part = int(number%10)
            frac = number - float(int_part)
            filenames.append("units_%d"%int_part)
            if abs(frac) > 1e-30:
                if tensSpecial=='BASIC':
                    return units[int_part]+u' ' + num2tamilstr(frac,filenames)
                elif tensSpecial=='NINES':
                    return units_suffix_nine[int_part]+u' ' + num2tamilstr(frac,filenames)
                else:
                    return units_suffix[int_part]+u' ' + num2tamilstr(frac,filenames)

            else:
                if tensSpecial=='BASIC':
                    return units[int_part]
                elif tensSpecial=='NINES':
```

```
                return units_suffix_nine[int_part]
            else:
                return units_suffix[int_part]
    else:
        if tensSpecial=='BASIC':
            filenames.append("units_%d"%number)
            return units[number]
        elif tensSpecial=='NINES':
            filenames.append("units_%d"%number)
            return units_suffix_nine[number]
        else:
            filenames.append("units_%d"%number)
            return units_suffix[number]

elif n_base == n_ten:
    if residue_number < 1.0:
        filenames.append("tens_%d"%(quotient_number-1))
        if residue_number == 0.0:
            return tens[quotient_number-1]
        #else: //seems not reachable.
        #    numeral = tens[quotient_number-1]
    elif number < 20:
        filenames.append("teens_%d"%(number-10))
        residue_number = math.fmod(number,1)
        teen_number = int(math.floor(number - 10))
        if residue_number > 1e-30:
            return teens[teen_number-1] +u' ' + num2tamilstr(residue_number,filenames)
        else:
            return teens[teen_number-1]+u' '
    if residue_number < 1.0:
        filenames.append( "tens_%d"%(quotient_number-1) )
        numeral = tens[quotient_number-1]+u' '
    else:
        if residue_number in tween:
            filenames.append( "tens_prefix_%d"%(quotient_number-2) )
            numeral = tens_prefix[quotient_number-2]
            tensSpecial='SPECIAL'
            if (quotient_number==9):
                tensSpecial = 'NINES'
        else:
            filenames.append( "tens_prefix_%d"%(quotient_number-2) )
            numeral = tens_full_prefix[quotient_number-2]+u' '
elif n_base == n_hundred:
    if residue_number == 0:
        filenames.append("hundreds_%d"%(quotient_number-1))
        return hundreds[quotient_number-1]+u' '
    if residue_number < 1.0:
```

```
            filenames.append( "hundreds_%d"%(quotient_number-1) )
            numeral = hundreds[quotient_number-1]+u' '
        else:
            filenames.append("hundreds_suffix_%d"%(quotient_number-1))
            numeral = hundreds_suffix[quotient_number-1]+u' '
    else:
        if ( quotient_number == 1 ):
            if n_base == n_thousand:
                filenames.append("one_thousand_prefix")
                numeral = one_thousand_prefix
            else:
                filenames.append("one_prefix")
                numeral = one_prefix
        else:
            numeral = num2tamilstr( quotient_number, filenames )
    if n_base >= n_thousand:
        suffix = suffix_base[n_base][int(residue_number >= 1)]
        suffix_filename = "%s_%d"%(suffix_file_map[n_base],int(residue_number >= 1))
        if residue_number == 0:
            filenames.append(suffix_filename)
            return numeral + u' ' + suffix+u' '
        filenames.append(suffix_filename)
        numeral = numeral + u' ' + suffix+u' '
    residue_numeral = num2tamilstr( residue_number, filenames, tensSpecial)
    #return numeral+u' '+residue_numeral
    return numeral+residue_numeral
# number has to be zero
filenames.append("units_0")
return units[0]
```

A.2 Parsing Tamil Number Words

```
def tamilstr2num(tokens):
    """
        இயல்மொழி எண்பகுப்பாய்வு.
        numeral parser; convert numeral to number.
        e.g. ["இருநூற்று","நாற்பத்தைந்து"] => 245
    """
    if isinstance(tokens,str):
        tokens = re.split(SPACE,tokens)
    is_american_str = False
    has_decimal = False
    US_values = [1e12, 1e9, 1e6]
    US_placements = ['டிரில்லியன்', 'பில்லியன்','மில்லியன்']
    IN_values = [1e7,1e7,1e5,1e5]
```

```
    IN_placements = ('கோடி','கோடியே','இலட்சம்','இலட்சத்து',)
    for tok in tokens:
        if tok in US_placements:
            is_american_str =True
        elif tok == 'புள்ளி':
            has_decimal = True
    value = 0
    stack = list()
    if is_american_str:
        HIGHEST=('டிரில்லியன்',)
        PLACEMENTS=US_placements
        VALUES=US_values
    else:
        HIGHEST=IN_placements[0:2]
        PLACEMENTS=IN_placements
        VALUES=IN_values

    for tok in tokens:
        if tok in PLACEMENTS:
            if len(stack) == 0:
                tmpval = 1.0
                if value != 0.0 and tok in HIGHEST:
                    value = value*VALUES[PLACEMENTS.index(tok)]
                    continue
            else:
                tmpval = helper_tamilstr2num(stack)
            stack = list()
            value += tmpval*VALUES[PLACEMENTS.index(tok)]
            continue
        stack.append(tok)
    if len(stack) != 0:
        value += helper_tamilstr2num(stack)
    return value

def helper_tamilstr2num(tokens):
    val_map = {}
    val_units = list(range(11))
    units = (u'பூஜ்ஜியம்', u'ஒன்று', u'இரண்டு', u'மூன்று', u'நான்கு', u'ஐந்து', u'ஆறு', u'ஏழு', u'எட்டு',
u'ஒன்பது', u'பத்து') # 0-10
    units_suffix = (u'பூஜ்ஜியம்', u'தொன்று', u'திரண்டு', u'மூன்று', u'நான்கு', u'தைந்து', u'தாறு', u'தேழு',
u'தெட்டு', u'தொன்பது', u'பத்து') # 0-10
    units_suffix_nine = (u'பூஜ்ஜியம்', u'றொன்று', u'றிரண்டு', u'மூன்று', u'நான்கு', u'றைந்து', u'றாறு',
u'றேழு', u'றெட்டு', u'றொன்பது', u'பத்து') # 0-10
    for _u in [units,units_suffix,units_suffix_nine]:
        for k,v in zip(_u,val_units):
            val_map[k]=v
```

```
    teens = (u'பதினொன்று', u'பனிரண்டு', u'பதிமூன்று', u'பதினான்கு', u'பதினைந்து',u'பதினாறு',
u'பதினேழு', u'பதினெட்டு', u'பத்தொன்பது') # 11-19
    for k,v in zip(teens,range(11,20)):
        val_map[k]=v
    tens = (u'பத்து', u'இருபது', u'முப்பது', u'நாற்பது', u'ஐம்பது',u'அறுபது', u'எழுபது', u'எண்பது',
u'தொன்னூறு') # 10-90
    for k,v in zip(tens,range(10,100,10)):
        val_map[k]=v
    tens_full_prefix = (u'இருபத்து', u'முப்பத்து', u'நாற்பத்து', u'ஐம்பத்து', u'அறுபத்து', u'எழுபத்து',
u'எண்பத்து', u'தொன்னூற்று') # 10+-90+
    for k,v in zip(tens_full_prefix,range(20,100,10)):
        val_map[k]=v
    tens_prefix = (u'இருபத்', u'முப்பத்', u'நாற்பத்', u'ஐம்பத்', u'அறுபத்', u'எழுபத்', u'எண்பத்',
u'தொன்னூற்') # 10+-90+
    for k,v in zip(tens_prefix,range(20,100,10)):
        val_map[k]=v
    hundreds = ( u'நூறு', u'இருநூறு', u'முன்னூறு', u'நாநூறு',u'ஐநூறு', u'அறுநூறு', u'எழுநூறு',
u'எண்ணூறு', u'தொள்ளாயிரம்') #100 - 900
    for k,v in zip(hundreds,range(100,1000,100)):
        val_map[k]=v
    hundreds_suffix = (u'நூற்றி', u'இருநூற்று', u'முன்னூற்று', u'நாநூற்று', u'ஐநூற்று', u'அறுநூற்று',
u'எழுநூற்று', u'எண்ணூற்று',u'தொள்ளாயிரத்து') #100+ - 900+
    for k,v in zip(hundreds_suffix,range(100,1000,100)):
        val_map[k]=v
    one_thousand_prefix = u'ஓர்'
    val_map[one_thousand_prefix] = 1.0
    thousands = (u'ஆயிரம்',u'ஆயிரத்து')
    val_map[thousands[0]] = 1000.0
    val_map[thousands[1]] = 1000.0
    one_prefix = u'ஒரு'
    val_map[one_prefix] = 1.0
    lakh = (u'இலட்சம்',u'இலட்சத்து')
    val_map[lakh[0]] = 100000.0
    val_map[lakh[1]] = 100000.0
    crore = (u'கோடி',u'கோடியே')
    val_map[crore[0]] = 10000000
    val_map[crore[1]] = 10000000
    pulli = u'புள்ளி'
    val_map[pulli] = 0.0

    mil = u'மில்லியன்'
    val_map[mil] = 1000000.0
    bil = u'பில்லியன்'
    val_map[bil] = val_map[mil]*1e3
    tril = u'டிரில்லியன்'
    val_map[tril] = val_map[bil]*1e3
    in_fractional_portion = False
```

```
multiplier = 1.0
value = 0.0



#["இருநூற்று","நாற்பத்தைந்து"] => 245
n_tokens = len(tokens)
for idx,tok in enumerate(tokens):
    n_remain = n_tokens - idx - 1
    if in_fractional_portion:
        multiplier *= 0.1
        value += multiplier*val_map[tok]
        continue
    if tok in mil:
        multiplier = 1e6
        continue
    elif tok in bil:
        multiplier = 1e9
        continue
    elif tok in tril:
        multiplier = 1e12
        continue
    elif tok in lakh:
        multiplier = 1e5
        continue
    elif tok in crore:
        multiplier = 1e7
        continue
    elif tok in thousands:
        multiplier = 1e3
        continue
    elif tok == pulli:
        if  multiplier > 1.0:
            if value > 0:
                value *= multiplier
            else:
                value = multiplier
        else:
            value *= multiplier
        multiplier = 1
        in_fractional_portion = True
        continue
    for _tens in tens_prefix:
         if tok.startswith(_tens):
             tok = tok.replace(_tens,'')
             value = value*multiplier + val_map[_tens]
             multiplier=1.0
```

```
            if tok != '':
                value += val_map[tok]
                tok = ''
            continue
        for _hundreds in hundreds_suffix:
            if tok.startswith(_hundreds):
                tok = tok.replace(_hundreds,'')
                value = (value)*multiplier + val_map[_hundreds]
                multiplier=1.0
                if tok != '':
                    value += val_map[tok]
                    tok = ''
                continue
        value = value*multiplier + (tok != '' and val_map[tok] or 0)
        multiplier=1.0
    if  multiplier > 1.0:
        if value > 0:
            value *= multiplier
        else:
            value = multiplier
    elif not in_fractional_portion:
            value *= multiplier
    return value
```