Journal of Science and Technology (JST)
Volume 2, Issue 2, March - April 2017, PP 22-27
www.jst.org.in

Language Expert Rendering Unicode Text On ASCII Editor For Indian Languages With Language Engine

A.Appa Rao
(Computer Science Department, Gunpur College of Engineering, India)

Abstract *: In this paper we introduce the Language Engine which addresses rendering of Unicode characters on a ASCII supportable editor. Unicode characters are related to Indian regional languages based on their character sets. With the application of Parse Engine, Language Engine and Rules Engine to identify the language and convert from Unicode to ASCII in the target editor to resolve rendering problem, also improvises file size reduction. Parse Engine parses the input text character by character and checks if it is a new text or already parsed. Inference Engine checks the Unicode character and identifies the language. Respective Language Engine is loaded with rules. Rules Engine identifies the corresponding ASCII text to be placed in output. Here Rules stored in Rules Engine are composed of XML formats and comes from manual entry. Input Unicode text can come from any source of input. Finally, the output composed from the output file is pasted on the ASCII editor with the installation of the font renders the correct text.*

Keywords - *Unicode, ASCII, Parse Engine, Language Engine, Rules Engine, Inference Engine*

I. INTRODUCTION

Machine learning is a branch of artificial intelligence science, the systems that can learn data. Artificial Intelligence has significantly gained grounds in our daily livelihood in this age of information and technology. As with any field of study, evolution takes place in terms of breakthrough or developmental research leading to advancement and friendly usability. Artificial Intelligence (AI) is the study of how to make computers (machines) do things Language Engine runs on the same lines of an inference engine. Inference Engine is a tool from Artificial Intelligence(AI). The first inference engines were components of Expert Systems. The typical expert system consisted of a knowledge base and an inference engine. The knowledge base stored facts about the world. The inference engine applies logical rules to the knowledge base and deduced new knowledge. This process would iterate as each new fact in the knowledge base could trigger additional rules in the inference engine.

Inference engines work primarily in one of two modes either special rule or facts: forward chaining and backward chaining. Forward chaining starts with the known facts and asserts new facts. Backward chaining starts with goals, and works backward to determine what facts must be asserted so that the goals can be achieved.

Provided with the regional ASCII fonts available. Fonts can be generated based on the requirement of the glyphs is discussed in the paper "Multilingual Font Creation by Mapping Unicode to ASCII". Forward chaining approach is selected as it starts with the known facts. Language Engine based on the rules of the language that appears in the text and with the provided rules for each regional language. In India we have our own regional languages , its indices and rules.

Rendering Problem is some editors does not show the Unicode character in its original form due to non supportable to utf-16. When u copy unicode text or data from any Input source and place on the editor they give question marks. Here they are called ASCII editors. For that reason, in order to solve that rendering problem we can go with Language Engine.

The introduction of the paper should explain the nature of the problem, previous work, purpose, and the contribution of the paper. The contents of each section may be provided to understand easily about the paper.

II. HEADING S FORWARD CHAIN RULES BASED SYSTEM

Input is the Unicode text and output is with the ASCII characters with its associated ASCII regional fonts based on the language. This is a unicode word looks like this

In Unicode supportable editor 
In Unicode unsupportable (only ASCII supportable – utf-8)

<0c24>+<0c46>+<0c32>+<0c41>+<0c17>+<0c41>.

Actually it looks like a three character word, but it has 6 unicode characters.

Each unicode character is 2 bytes, 12 bytes of space is occupied if it is in memory.

We can minimize this by the following procedure:

1. Identifying the Unicode characters to which language set it belongs to.
2. By mapping Unicode characters to ASCII (Creation of a new font or select the existing regional language ASCII font - vendor)
3. Production of rules form that font.
4. Rules are composed based on its language and its meaning etc.,

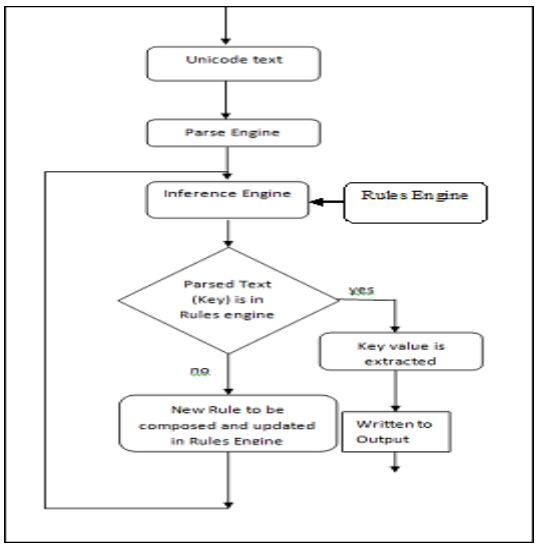
These rules will help in arranging the translated words correctly based on the context of the sentence. This rule based system consists of various steps like syntax analysis, semantic analysis, morphological analysis, syntax generation and semantic generation. Rule based systems are less robust and gives good grammatical results if it finds an appropriate parse else it fails. Here fails means the rule does not exist. So a new production rule has to be added.

Rule based methodologies can be broadly classified as direct, transfer and Interlingua [9]. In direct methodology, there are no intermediate stages in the translation. It doesn't use any complex rules or parsing structures. This method makes use of syntactic and semantic similarities of source and target languages. Transfer methodology works in three phases namely analysis, transfer and generation. Transfer method consists of complex rules. Interlingua method works in two phases. The source text is converted into an Interlingua representation from which the text in target language is generated. Based on the glyph location in the ASCII set production rule has to be designed. User Interface tools are available to produce or create the rules based on the fonts. At the time of parsing of text every character is taken with its Unicode value and a enclosed in tag format and a delimiter + is added, in order to be easy for matching the rule for its presence in the provided resource.

Placing the required language (Telugu) characters with its sets in the ASCII set along with its vowels, consonants, numbers etc. Selecting that characters it becomes three letter word. Facts are dumped in the Rules Engine. When a hit occurs it selects the fact matching rule from rules engine and places that characters in the output. In this way we can parse word by word. Once the hit is occurred we can place that in our database as not to parse once again. Already we have parsed it and the rule is available. Resources are to be placed in the projects Resources folder. Fonts, Rules, Languages are some of the resources. Fonts can be TTF, OTF. For ASCII based font insertions we use .PFA(post script ASCII font) files. For binary based we use PFB(post script binary font) files. Rules resources, language sets resources are to be provided in xml formats which is easy for editing , updating.

Rules Engine

Rules are xml based elements with key value pair. Key can have combination of characters and its extensions as well as value can have combination of characters with extensions. But the value of the character changes from Unicode to ASCII



Flow of Language Engine

Rules generally prepared by language experts of that regional. Here the input and output both are of same language. There will not be any difference in the output means visual display. But changes in the rendering format and file size it consumes.

Parse Engine

Parse Engine will take input from the document word by word. Extracts the unicode range value and identifies the character set. Languages.xml file contains the language ranges that are specified in the unicode.org. Here we are restricting to Indian Regional languages.

Snapshot of Languages.xml file

```
<Languages>
<Tamil>0B80-0BFF</Tamil>
<Telugu>0C00-0C7F</Telugu>
<Kannada>0C80-0CFF</Kannada>
….
</Languages>
```

Identifies the unicode language by checking the character from Language.xml file. The related Language file is found in Resources folder and is loaded in to memory.

Parser will parse the text and checks in the database , if the hit already done or a new hit. If it doesn't matches , it passes to Rules Engine. Rules Engine checks the match for first unicode character and if it matches , then next character like that tries to match for a rule with all the characters sequentially.

Rules Engine: It is an XML based DOM file with each language rules in its name.

Telugu language - Telugu.xml

Kannada language - Kannada.xml

…

```
Telugu.xml
<Telugu "Range"="0C00-0C7F">
<Rules>
<0c24>+<0c46>+<0c32>+<0c41>+<0c17>+<0c41>
=<0040>+<0052>+<0035>
...
</Rules>
</Telugu>
```

Once the Language file is loaded in to memory, it forms a dictionary with key value pairs. Keys will be unique. In the above Rules the left side of '=' is key and right side is value. Rules are loaded in to Map or Dictionary. The parsed text that has been given to Rules Engine as input is matched with the keys, if it matches the equivalent value is extracted and given back as output to Parse Engine.

Inference Engine

At the time of parsing of text if the parsed text match is found in the database it infers that already existing, directly it provides the parser with a value, else it infers the Rules Engine to check for a match. If the rules engine did not find a rule from the loaded resource , then there is a facility to add or compose a new rule for a new word that is not found.

To compose Production rules(facts) for Telugu language below are some rules

Note

x, **X** and ` [character under '~' in English keyboards] are used as modifiers and/or splitters

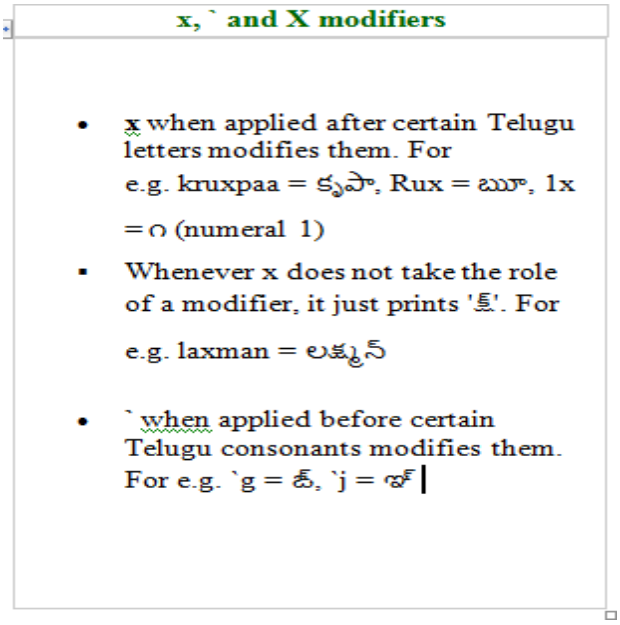
Vowels
అ = a, ఆ = aa/A, ఇ = i, ఈ = ee/ii/I, ఉ = u, ఊ = oo/uu/U ఎ = e, ఏ = ae/E, ఐ = ai, ఒ = o ఓ = oa/O, ఔ = au/ou/ow అం = aM/amx, అః = aH ఋ = rux, ౠ = Rux,

క్ + vowels
క = ka, కా = kaa/kA, కి = ki, కీ = kee/kii/kI, కు = ku, కూ = koo/kuu/kU కె = ke, కే = kae/kE, కై = kai, కొ = ko, కో = koa/kO, కౌ = kau/kou/kow కం = kaM/kamx, కః = kaH కృ = krux, కౄ = kRux

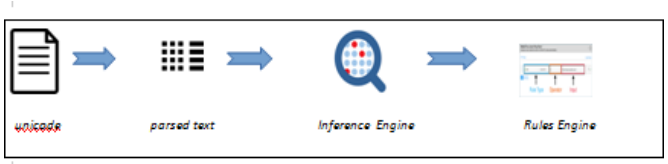
Consonants	
క = ka, ఖ = Ka/kha, గ = ga, ఘ = Ga/gha, ఙ = `ga చ = ca/cha, ఛ = Ca/Cha, జ = ja, ఝ = Ja/jha, ఞ = `ja ట = Ta, ఠ = Tha, డ = Da, ఢ = Dha,	

ణ = Na త = ta, థ = tha, ద = da, ధ = dha, న = na ప = pa, ఫ = Pa/pha, బ = ba, భ = Ba/bha, మ = ma య = ya, ర = ra, ల = la, వ = va/wa శ = sha/Sa, ష = Sha, స = sa, హ = ha ళ = La, ఱ = Ra, క్ష = ksha/xa, జ్ఞ = j`ja	

Numerals
౧ = 1x, ౨ = 2x, ౩ = 3x, ౪ = 4x, ౫ = 5x ౬ = 6x, ౭ = 7x, ౮ = 8x, ౯ = 9x, ౦ =0x



Expert system = knowledge base + Inference Engine
Language Expert is a tool that has knowledge of language in the form of base rules and production rules. Base rules are with vowels, consonants, numerals. Production rules are compositions of consonants with vowels and some modifiers. Similarly this can be done for other regional language with its rules and specifications.



Control Flow Diagram

At the parse engine level, and at rules production it takes the above provided specifications for telugu language. For some consonants all vowels are not framed. With its frequent occurrences the combinations are mapped in the fonts. In this way in the forward chaining we can make a language engine which will be useful for conversion of unicode to ASCII.

III. CONCLUSION

We have briefly described how to convert Unicode text to ascii for the rendering of the editors that doesn't support Unicode characters Vendor creates fonts for required data sets where size of the file is the criteria to be concentrated. This paper is concentrating and giving an overview of how to create rules for a regional font with its rules framing. This can be extended further for other languages also. It may include CJK (Chinese-Japanese-Korean) languages. But due to more character sets its space consuming.

REFERENCES

[1] Peter J.F. Lucas & Linda C. van der Gaag "Principles of Expert Systems", Centre for Mathematics and Computer Science, Amsterdam, published in 1991 by Addison-Wesley
[2]Akshar Bharati, Vineet Chaitanya, Amba P. Kulkarni and Rajeev Sangal, " ANUSAARAKA: Machine Translation in Stages", A Quaterly in Artificial Intelligence, Vol. 10, No. 3, July 1997

[3] Latha R Nair and David Peter S, "MAchien Translation system for Indian Languages", IJCA, Vol 39, No. 12012
[4] http://www.azhagi.com/az-telugu-classic.html
[5]Sugata Sanyal and Rajdeep Borgohain, "Machine Translation Systems in India", arXiv, April 2010.
[6] Lewis, M. Paul, Gary F. Simons, and Charles D.Fennig(eds.). 2014. Ethnologue: Languages of the world, seventeenth edition, Dallas Texas: Sil International. http://www.ethnologue.com.
[7] Antony, P. J. "Machine Translation Approaches and Survey for Indian Languages." *Computational Linguistics and Chinese Language Processing Vol* 18 (2013): 47-78.

[8] http://www.unicode.org/versions/Unicode8.0.0/UnicodeStandard-8.0.pdf

[9] https://www.cl.cam.ac.uk/~mgk25/unicode.html

[10] Antony, P. J. "Machine Translation Approaches and Survey for Indian Languages." *Computational Linguistics and Chinese Language Processing Vol* 18 (2013): 47-78.

[11] Antony P J and Dr. Soman K P Machine Transliteration for Indian Languages: A Literature Survey International Journal of Scientific & Engineering Research, Volume 2, Issue 12, December-2011 1 ISSN 2229-5518