

HINDAWI – A SOFTWARE DEVELOPMENT SYSTEM FOR NORTH INDIAN VERNACULARS

Abhishek Choudhary AMCSI, B.B.A., M.A. Linguistics, **Sweta Karwa**

Indicybers (A division of Anamika Press, Katihar), K-6, Tollygunge Police Lines, Kolkata – 700 033
Phone: 24221175(telefax)/9831369549 <http://www.indicybers.com/> Email: choudhary@indicybers.net

Introduction and objectives of the project

The promises of India becoming an IT superpower in the near future shall bear fruit only if it takes into account the majority of the populace which has not been privileged enough to have mastered English, which in turn hinders them from participating in the larger picture of Information and Communication Technologies (ICT) owing to the lack of software development systems based on the vernaculars. This paper presents the design and development of Hindawi – a complete software development (programming) system for the North Indian vernaculars and answers many “key” technological issues that hinder the development of such a system.

The authors have dealt with the issue at all levels of systems design, having successfully developed technologies for console representation of North Indian scripts in “text-mode” without the need for any extra hardware and a method for compilation of ISCII based source code. The complete glyphs for the North Indian scripts have been successfully incorporated into extended ASCII code pages while maintaining the 7-bit ASCII unaltered. The representation also allows easy integration into UNICODE. The compilation of ISCII based source code has been achieved through the design of a non-ambiguous invertible case and diacritic independent transliteration system compatible with most compiler systems.

The use of GNU GCC as the back end has ensured high portability across a heterogeneous range of platforms. The authors have successfully developed programming languages based on most prevalent paradigms including imperative, OOP, functional, and logic among others; leading to the development of the equivalents of at least Assembly, BASIC, C, C++, FORTRAN, Pascal, LISP, Prolog, lex, and yacc in North Indian vernaculars, in languages using the Assomiya, Bangla, Devnagri and Gujrati scripts, with the Oriya and Gurmukhi scripts under development. This has made possible the porting of the LINUX kernel sources to the Hindawi programming system. A port to FreeDOS is also being developed. The authors’ expertise in robotics has allowed them to apply Hindawi to robotics systems design. The system promises to be able to support South Indian languages soon.

The authors have made the complete system, including the design of a North Indian vernacular understanding robot, “Open Source” in keeping with the primary objective of providing India’s “vernacular literate” a voice in the race towards making India an IT superpower.

Contributions of the project

The plethora of software being developed for the Indian vernaculars is generally focussed towards the application domain with very little or no emphasis on Indian languages system-software, with the exception of a few “shell wrappers”. This has led to the situation where the majority of the populace do have an opportunity to become consumers of the products of the ICT revolution, but cannot become contributors to it, in any significant manner. The reason for this may be traced to a few technological bottlenecks, most of which have been answered during the course of this project.

The primary bottleneck has been that of compilation of Indian script source code, as there is no systems level support for the same. The ISCII puts the Indian codes in the extended half of the 8-bit code pages, which are not supported by most compiler systems. A solution to this is the use of a transliteration layer between the source and the language processor, however existing transliteration systems produce output incompatible with the input set of the processors, which in most cases introduces ambiguity upon inversion. The authors have been successful in developing a phonetically sound invertible non-ambiguous Roman script transliteration system that produces output compatible with most compilers thorough case and diacritic independence. This has allowed the development of programming languages for major paradigms in most North Indian vernaculars and allows Indian language URLs inherently.

A second bottleneck has been the inability to display Indian scripts in “text mode”. The existing Indian languages software generally operates in graphics mode or makes use of extra hardware, which adds to the cost of procurement. The authors have succeeded in developing a method for “smart” rendering of North Indian scripts in text-mode without the need for any extra hardware components. The semantics for Assomiya, Bangla, Devnagri and Gujrati scripts have been completed, while those of Oriya and Punjabi are at an advanced stage of development. This

has allowed the gap between machine code and graphics development environments to be bridged in North Indian vernaculars, including machine opcode assembly, OS kernel and command shell. The glyphs have been comfortably accommodated in the extended ASCII code-pages with unaltered 7-bit codes. The rendering quality exceeds or at least equals that achieved in most graphics-mode systems.

The bottom-up approach of the project has allowed the authors to maintain uniformity, in selection of tokens (keywords) for matching language constructs across programming paradigms. This promises to increase the gradient of the learning curve for trainees in North Indian vernacular based systems development (programming), as the barrier posed by learning new keywords for increasingly complex paradigms is reduced to negligible.

A certain level of “literate” programming with support for machine translation of documentation across North Indian vernaculars and English has also been achieved. This allows for programming tasks specified in English to be handled by personnel trained in vernacular programming, with the output product and documentation being acceptable internationally.

The incorporation of higher-level paradigms shall allow the target populace to contribute to the ICT revolution in more ways than the task of simple application development. Logic and functional paradigms shall allow work in Artificial Intelligence to be performed in North Indian vernaculars. The authors’ have also applied the system to the development of an intelligent natural-interfaced robotics system.

The complete system, including the design of the Indian vernacular understanding robot, has been made “open-source”, with the target of meeting the financial objectives through support services. This shall allow low procurement costs for Indian vernacular development systems, and make the maintenance of the programming system a community effort. This shall also allow the system to be used for pedagogical purposes.

Methodology and algorithms

The algorithms developed during the course of the project are enumerated below, followed by a description of the methodology used for developing programming systems in North Indian vernaculars.

1) Romenagri - a non-ambiguous invertible case and diacritic independent compiler acceptable transliteration system with the associated algorithm has been the sole basis for much of the project’s motivation. The authors have independently developed it and demonstrated it to be applicable to all languages using the North Indian composite syllabic scripts – viz. Assomiya, Bangla, Devnagri, Gujrati, Oriya and Punjabi. Romenagri utilises syllabic complements in Roman script for the symbols of the North Indian

scripts. The mapping for a specific script may be a subset of the complete mapping owing to the absence of certain characters in the specific case, e.g. the *wa* and *ba* of Devnagri match a single symbol in Bangla *ba*. The words are formed by actively concatenating successive syllabic compliments, looked up from a table through an O(1) search achieved by using the normalised code for the Indian script symbol as an array index. The process of active concatenation uses the “de-voweling” operator carat (^), which forms an equivalent of *halanta* or *hasanta* of the Indian scripts and distinguishes the *matra* of the vowels by preceding the syllabic compliment of their *akshara* form. The syllabic compliment looked up from the mapping table is pushed onto a stack. On encountering a carat as part of a looked-up compliment, the last pushed vowel character “a” is popped out of the stack and discarded. The remaining part of the compliment, after the carat, is then pushed onto the stack. On encountering the end of a word, the content of the stack is popped to obtain the required transliteration, after which the stack is flushed.

The process of converting Romenagri back to the Indian script representation is more complex and is achieved by using a recursive descent parser. The authors have designed the syllabic compliment so as to facilitate O(log n) parsing. The parser operates at 5 levels. The word is submitted at level 1, and the initial syllabic compliments are consumed. Successive levels are entered in case of multiple possibilities with the ultimate level identifying a *matra*. All other symbols are identified at earlier levels. After each production the parser enters level 1 with the non-consumed part of the input.

The only phonetic modifier used in Romenagri is the underscore “_” character, which generally forms a part of the input set of most compilers. This allows rule adherent transliteration for keywords written in Indian scripts. The underscore characters present in the original Indian script text are expanded to two underscore characters. Hence, the inversion parser treats every paired underscore as a character and every nascent underscore as a phonetic modifier. Figure-1 gives an instance of Romenagri transliteration.

$$\text{क + ्र + ि + य + ा = क्रिया}$$

$$ka + ^ra + ^i + ya + ^aa = kriyaa$$

Figure 1: An instance of Romenagri transliteration with corresponding syllabic compliments

2) APCISR (Anamika Press Code for Indian Script Representation) is a complex of representational semantics for compositional syllabic Indian scripts along with the corresponding set of graphemes, developed independently by the authors, for use with fixed width console (text-mode) applications. The APCISR uses a 9-grid format to extract the common features of the Indian script symbols as instantiated in Figure-2. Each feature forms a specific grapheme. The

9-grid consists of three rows, viz. *Urdha*, *Madhya* and *Nimna*, and three columns, viz. *Matrik*, *Lipik* and *Purak*. The Indian script symbols are mapped to their constituent graphemes in one table, with the graphemes being mapped to the corresponding glyphs (character-codes) in another table. Hence, the process of conversion of codes such as ISCII to APCISR is a two-step procedure. The first step (synthesis) consists of combining the grapheme maps of the different Indian symbols, which is algorithmically intensive, while the second step is a straightforward O(1) lookup procedure for obtaining the character values of the corresponding graphemes.

The explanation of the synthesis step requires us to distinguish between the look-up map (LM) and the working-map (WM). The LM is a simple 9-grid grapheme map, while WM consists of three rows of three or more columns, with three cursors pointing out the *Matrik*, *Lipik* and *Purak* columns, each of which can move independently with respect to each other. The LM grapheme maps also contain other related properties of the Indian script symbols, such as how the incorporation of the LM in the WM moves the cursors of the WM. This forms a basis for a set of semantic rules for the synthesis step, such as upon encountering a half consonant the *Matrik* remains constant while the *Lipik* and *Purak* are right shifted by 1 place, making the previous *Purak* the current *Lipik* and introducing a new column to the right of the WM, which becomes the new *Purak*. A normal consonant cursor shift consists of the existing *Purak* becoming the new *Matrik*, along with the introduction of two new columns to the right of the WM for the new *Lipik* and *Purak*. A normal *matra* causes no cursor shift. The LM grapheme table also consists of mappings for character combinations (*sanyuktakshara* or *juktakshara*), which are treated as a single symbol. Once the position of the cursor has been determined, the LM values are logically AND-ed with the corresponding WM values. However, some scripts deviate from generalisations and require the inclusion of specific rules, which are economically accommodated at the end of the synthesis step. The process of APCISR conversion is reversible, however the step of character-code rendering introduces some ambiguity prohibiting proper reconstruction, owing to the fact that more than one grapheme may use the same character code. This issue can be addressed by using larger character pages. However, as our objectives do not require the APCISR to be reversible, as the rendering is done just-in-time, and using the same character-code or glyph for different graphemes allows the extended ASCII code-page to accommodate the glyphs while maintaining common graphic symbols such as box and shaded bars, with the 7-bit code page remaining constant. The conversion of the standard international numerals to the corresponding script has been kept optional. APCISR is currently supported on VGA and compatible graphics adapters.

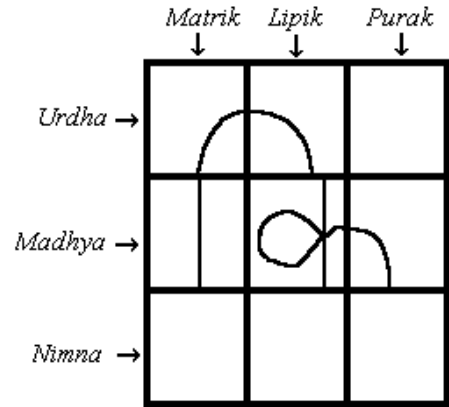


Figure 2: 9-grid fixed width representational semantics for North-Indian scripts

Hindawi includes a rudimentary support for “literate” programming, where documentation is generated from the comments contained in the source code, along with machine translation support for a restricted language subset. This has been achieved by allowing only specific sentence constructions of the SVO form. This is parsed using a *yacc* grammar followed by a dictionary-based translation. This method allows for the objective of easily convertible documentation to be met, but restricts expressiveness.

The methodology used by the authors for generating various compilers using Romenagri and APCISR consists of writing lexical analysers in *lex* for translating keywords between the Indian vernacular programming language and a matching programming language in English. Specific language constructs have also necessitated the writing of *yacc* parsers, especially where semantic changes are necessary. The keywords for matching programming constructs have been kept constant across paradigms. Table-1 lists the common paradigms and their Hindawi equivalent. Table-2 lists the various keywords used for the common programming constructs in Hindawi.

<u>Programming paradigm</u>	<u>Hindawi shaili</u>
1) Beginners	1) <i>Prathmika</i>
2) Imperative	2) <i>Guru</i>
3) Object Oriented	3) <i>Shraenibadha</i>
4) Logic	4) <i>Taarkika</i>
5) Functional	5) <i>Kriyaatmaka</i>
6) Analyser generators	6) <i>Shabda</i>
7) Parser generators	7) <i>Vyakarana</i>

Table 1: Common programming paradigms and their equivalent Hindawi *shailis*

For	<i>krama</i>	write	<i>likho</i>
While	<i>jaba tak / jata khon</i>	read	<i>poocho / prashna</i>
If	<i>agar / jyodi</i>	begin	<i>Anu</i>
File	<i>khata</i>	End	<i>purna</i>

Table 2: Common programming constructs and their Hindawi equivalent for Hindi

GNU GCC has been used as the backend for almost all *shailis* (programming languages) of Hindawi, allowing portability to heterogeneous platforms. A driver controls the compilation procedure calling various filters and processors alternatively, in a manner similar to that performed by *gcc*. The specific order of the calls made by the driver is given below.

- a) The source code in ISCII is passed through a filter performing the Romenagri transliteration.
- b) The Romenagri source code is then passed through a *shaili*-specific compiler to obtain *gcc* compatible source code.
- c) A final call is then made to *gcc* or *g++* depending upon the *shaili*.

The authors have also developed *Lekhak*, an IDE for Hindawi, which also doubles as an Indian language word-processor supporting search, cut, copy, paste etc. along with an online help system for Hindawi programming. Hindawi supports graphics under DOS through the Allegro graphics system and support for printing on PCL-compatible printers has been independently developed by the authors. The authors have also enabled Hindawi to be used for robotic systems design and have successfully developed an Indian-language understanding robot using it. A command shell for DOS and Linux has been developed using Hindawi and effort is being put into the early completion of the translation of Linux and FreeDOS kernels to Hindawi.

Financial and social feasibility

The financial objectives of Hindawi have been targeted to be met through support services and licensing of non-GPL parts for commercial development. Demonstrations may be downloaded from the authors' website. CD-ROM version of the complete system is made available on request at cost of material and posting only.

The complete system along with the design of the Hindawi based intelligent robot has been made "open-source". This has been done with the primary social objective of tapping into the vast "vernacular-literate" manpower potential in ICT by bridging the language barrier. The provisions for graphics and sound support even at beginners' level promise to make Hindawi popular in a manner similar to the popularity BASIC has gained. The support for higher programming paradigms allows for serious tasks to be based upon Hindawi. The inbuilt "literate" programming and machine translation of documentation, though rudimentary, allow an international feasibility for Hindawi-based development. Open-source software essentially becomes a community maintained product. The authors expect Hindawi's capabilities and popularity to increase with increased effort in its development. The copyright for Hindawi, however, has been retained by the authors' organisation, in conformance with GNU General Public Licence V2, in parts that utilise GPL

software and only these parts of Hindawi come under GPL, with the sole motive of preventing the development process losing its primary objective. Hindawi, including non-GPL parts, is free for pedagogical purposes.

Conclusion and further work

A note on the selection of the title "Hindawi" is deserved before concluding. The word Hindawi is of Arabic origin, used to describe "all" regions, their people and languages across the Indus River. The term finds earliest references in the works of Amir Khusro, the renowned poet. The authors chose this word as it is the most appropriate word that may be used to denote "all" the languages of India.

The Hindawi project has yielded many technological benefits such as APCISR and Romenagri along with the feasibility of using vernacular programming at all levels of systems development. The project is currently in a stable state, but may indeed be considered to be in the infancy of its final goal. Immediate development requirements include completion of representational semantics for the remaining Indian scripts and language mappings for remaining vernaculars. Finally, the authors expect corporates and other financially sound organisations, interested in activities of nation building, to come forward and make an effort towards spreading ICT involvement through vernacular programming. Interested individuals may lend their support through participating in the further development and spread of vernacular programming.

Appendix

ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
128	129	130	131	132	133	134	135
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
136	137	138	139	140	141	142	143
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
144	145	146	147	148	149	150	151
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
152	153	154	155	156	157	158	159
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
160	161	162	163	164	165	166	167
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
168	169	170	171	172	173	174	175
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
176	177	178	179	180	181	182	183
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
184	185	186	187	188	189	190	191
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
192	193	194	195	196	197	198	199
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
200	201	202	203	204	205	206	207
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
208	209	210	211	212	213	214	215
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
216	217	218	219	220	221	222	223
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
224	225	226	227	228	229	230	231
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
232	233	234	235	236	237	238	239
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
240	241	242	243	244	245	246	247
ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ	ॐ
248	249	250	251	252	253	254	255

Figure 3: Glyphs for APCISR graphemes of Devnagri accomodated in 8-bit extended ASCII