PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 15/20

A1

(11) International Publication Number: WO 94/25922

(43) International Publication Date: 10 November 1994 (10.11.94)

(21) International Application Number: PCT/US94/00081

(22) International Filing Date: 3 January 1994 (03.01.94)

(30) Priority Data:
08/053,790 26 April 1993 (26.04.93) US

(71) Applicant: TALIGENT, INC. [US/US]; 10201 N. de Anza Boulevard, Cupertino, CA 95014 (US).

(72) Inventors: DAVIS, Mark, E.; 10174 Potters Hatch Common, Cupertino, CA 95014 (US). LIN, Judy; 1485 De Rose Way, #222, San Jose, CA 95126 (US).

(74) Agent: STEPHENS, Keith; Taligent, Inc., 10201 N. de Anza Boulevard, Cupertino, CA 95014 (US).

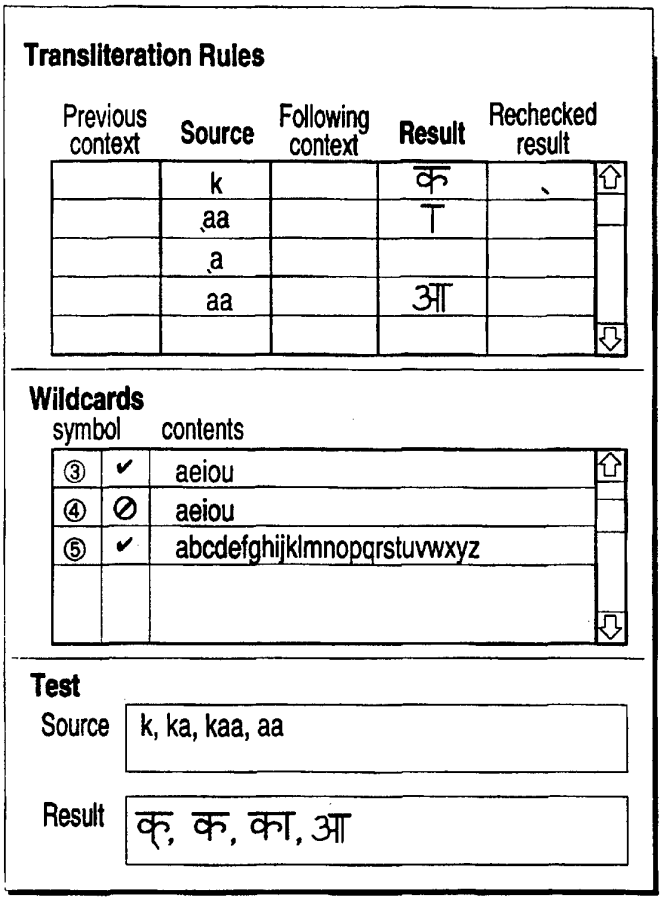
(81) Designated States: AT, AU, BB, BG, BR, BY, CA, CH, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, KZ, LK, LU, LV, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SK, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: TEXT INPUT TRANSLITERATION SYSTEM

(57) Abstract

A method and system for providing text input transliteration processing is disclosed. An innovative system and method for performing the transliteration is presented that performs processing on text as it is input to a computer.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

TEXT INPUT TRANSLITERATION SYSTEM

COPYRIGHT NOTIFICATION

Portions of this patent application contain materials that are subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

CROSS-REFERENCE TO RELATED PATENT APPLICATIONS

This patent application is related to the patent application entitled Text Transliteration System, by Mark Davis and Judy Lin, filed concurrently with this application, and assigned to Taligent, the disclosure of which is hereby incorporated by reference.

Field of the Invention

This invention generally relates to improvements in computer systems and more particularly to intelligently transliterating text as it is input to a computer system.

Background of the Invention

US patent 5,148,541 discloses a multilingual database system including sorting data using a master universal sort order for all languages. The database system can be searched and retrieved by a user whether or not that data is in the user's own language. The data to be stored in the database is first encoded according to a master (or universal) sort order.

US patent 4,734,036 discloses a method and device for learning a language. The patent discusses a teaching aid for reinforcing a student's ability to learn an unfamiliar language including an upper sheet (12) marked with symbolic indicia to be taught to the student and one or more base sheets (11), each marked with a different translated version of the indicia on the upper sheet. The indicia on each base sheet are marked in registry with the corresponding indicia on the upper sheet. One edge of the base sheet is joined, temporarily or permanently, to a corresponding

edge of the upper sheet to allow the upper sheet to be lifted up from the base sheet to briefly expose a corresponding translation, transliteration, interpretation, or paraphrase marked on the base sheet then lowered again so that reading of the upper sheet can be instantly resumed.

US patent 4,547,765 discloses a method and circuit arrangement for transliteration of code words of a code having m-place code words into corresponding code words of a different code likewise having m-place code words, individual bits of the code word to be translitereated are forwarded during serial input into a m-place shift register or during the serial output therefrom. These bits are forwarded non-negated or negated from register stage to register stage over a respective forwarding circuit depending upon the measure or criterion of coincidence or non-coincidence between the code word to be transliterated and the code words of the different code. This occurs in such manner that the traversing bits experience a respective negation in front of and after a register stage whose position within the shift register corresponds to the position of non-coinciding bits within the two code words.

Systems such as the Apple ® Macintosh ® or Microsoft ® Windows (TM) have dead keys which are employed to extend the range of the keyboard for accented characters. With this mechanism, a user can type a key (e.g. option-u for umlaut) which puts the keyboard into a special state, but does not generate a character or any other visible indication of what has occurred. When the user then types a base character—one that combines with the accent—then the keyboard generates the resulting accented character, for example, typing option-u, e produces ë). However, this approach requires a user to be cognizant of particular special keys associated with a particular task.

Summary of the Invention

Accordingly, it is a primary objective of the present invention to provide a set of flexibly defined rules stored in data structures in a computer system to automatically apply user specified transliterations to text as it is input to a computer system.

Brief Description of the Drawings

Figure 1 is a block diagram of a personal computer system in accordance with a preferred embodiment;

Figure 2 is a flowchart of the logic used to transliterate text between offsets start and finish in accordance with a preferred embodiment;

Figure 3 is a flowchart of the logic used to identify a type-in starting point in accordance with a preferred embodiment;

Figure 4 is a flowchart of the logic used to provide type-in transliteration in accordance with a preferred embodiment;

Figure 5 is an illustration of a display in accordance with a preferred embodiment; and

Figure 6 is an illustration of a transliteration operation as it would appear on a user's display in accordance with a preferred embodiment of the invention.

Detailed Description Of The Invention

The invention is preferably practiced in the context of an operating system resident on a personal computer such as the IBM ® PS/2 ® or Apple ® Macintosh ® computer. A representative hardware environment is depicted in Figure 1, which illustrates a typical hardware configuration of a workstation in accordance with the subject invention having a central processing unit 10, such as a conventional microprocessor, and a number of other units interconnected via a system bus 12. The workstation shown in Figure 1 includes a Random Access Memory (RAM) 14, Read Only Memory (ROM) 16, an I/O adapter 18 for connecting peripheral devices such as disk units 20 to the bus, a user interface adapter 22 for connecting a keyboard 24, a mouse 26, a speaker 28, a microphone 32, and/or other user interface devices such as a touch screen device (not shown) to the bus, a communication adapter 34 for connecting the workstation to a data processing network and a display adapter 36 for connecting the bus to a display device 38. The workstation has resident thereon an operating system such as the Apple System/7 ® operating system.

KEYBOARD TRANSLITERATORS

On the Apple Macintosh computer, dead keys are used to extend the range of the keyboard for accented characters. With this mechanism, a user can type a key (e.g. option-u for umlaut) which puts the keyboard into a special state, but does not generate a character or any other visible indication of what has occurred. When the user then types a base character—one that combines with the accent—then the keyboard generates the resulting accented character (e.g. option-u, e produces ë).

Dead-Key Example

Key	Deadkey state	Display
b	<none>	b
option-u	<umlaut>	b
a	<none>	bä
d	<none>	bäd

In a preferred embodiment of this invention, the modal mechanism is replaced by the use of transliterators. When an application inserts characters from a keyboard into some text, then it will call the list of transliterators associated with that keyboard, and the input method for the keyboard. An accent transliterator can provide the same functionality as dead keys. When an accent is typed, it will be combined with the previous "base" character if possible. For example:

Key	Pre-transliterate	Display
b	b	b
a	ba	ba
option-u	ba¨	bä
d	bäd	bäd

Transliterators also perform many other functions. For example, they can replace generic quotes (",') by righthand and lefthand quotes (“,‘,’,”). They can also be used to perform general script transcriptions for any cases where the transcription is simple and unambiguous, as when converting from romaji to katakana or hiragana for japanese, converting jamo (letter components) to hangul (letter syllables) for Korean, or converting Querty to Hebrew, etc. By convention, an apostrophe can be used to prevent characters from being transliterated together. For example, to form "ba¨ d", one would type "ba'¨d".

INPUT TRANSLITERATION

Transliteration can also be used to phonetically convert between different languages. This feature is especially important for languages such as Japanese that use a Roman keyboard to key in text, which is then transcribed into native Japanese characters. These characters can also be converted back into Roman characters. A particular class of transliterations called *input transliterations* obey two requirements set forth below.

- Uniqueness
  Transcription from native to foreign script is unambiguous. Two different native strings cannot correspond to the same foreign string. For example, if the native script distinguishes between a retroflex and a dental T, then a transliteration cannot map them onto the same symbol “t”.

- Completeness
  Transcription from native to foreign script, or from foreign to native is complete. Every sequence of native symbols will map onto some string of foreign symbols. Transcription from foreign to native script should be complete, but is not generally unambiguous. For example, when a Roman-to-Japanese transcription is used, “ra” and “la” map onto the same Japanese symbol.

A TTransliterator object is used to perform transliterations. Input transliterators are composed of a set of context-sensitive rules. These rules are designed to allow non-programmers to edit them reasonably for localization.

Examples of rules:

cho => ちょ

t[t => っ
to => と

Using these rules, chotto can be transliterated into: ちょっと.

Transliteration may be dependent not only on script but also on language. It is also inherently an *n* x *n* problem: expecting to transliterate from Russian to Hindi by using a series of Cyrillic-Roman and Roman-Hindi transliterations is doomed to failure, since the transcriptions used to represent non-Roman letters will vary depending on the script being represented: in some cases *th* will represent the sound found in *thick*, while in others it is an aspirated *t*.

A preferred embodiment provides input transliteration from Roman to Japanese (Hiragana and Katakana), Russian, Greek, Arabic, Devanagari (Hindi), and Hebrew. There is also a "Symbols" transliterator which allows a user to enter any Unicode symbol by name i.e.., "apple-logo", and transcribe it to the actual character, .

Transliterators can be chained. For example, one may wish to have a "smart-quote" transliterator be the first in chain, followed by an input transliterator. This mechanism is managed by the TTypingConfiguration object.

TEXT TRANSFORMATION THROUGH TRANSLITERATION

Transliteration can also be used for language-specific processing such as converting text from upper to lower case and back, creating title text. For example, "With the Important Words Capitalized"; title text requires use of a title-filter text service in order to eliminate articles. Otherwise, the titled text will appear as: "With The Important Words Capitalized."), and stripping diacritical marks. Note that these results are achieved by modifying the text objects directly. If the transformation is only desired for display, and should not effect the actual text objects, there is an alternative method employed using metamorphosis tables in the font.

TRANSLITERATOR DESKTOP OBJECTS

Transliterators are desktop objects. For example, a user would add a transliterator to his typing configuration by dragging it from the desktop to a typing configuration encapsulator. *TTransliteratorModel* encapsulates, accesses and

manages the transliterator data. *TTransliteratorUserInterface* is responsible for the icon and thumbnail presentations as well as the user interface to edit the data. A *TModelSurrogate* is used as a stand-in for the model, and will typically be imbedded in the typing configuration model.

Programmatic access to all available transliterators: TTransliterator::GetAvailableTransliterators(TCollection&) that will return a collection of TModelSurrogate objects.

Identifying transliterators: Transliterator objects are stored as TFiles. The Pluto attribute is used to identify a transliterator.

Transliteration Internals

Background

Rule-based transliteration is designed for relatively unsophisticated users and localizers to be able to create and modify. The rules are designed to be straightforward, and to apply especially to the case of transliteration as the user types. Although designed to meet the specific needs of transcribing text between different scripts, either during user type-in or converting a range of text in a document, transliteration uses a general-purpose design which is applicable to a wide range of tasks.

A transliteration rule consists of two main parts: a source and a result. The source can be accompanied by two strings which specify the context in which the conversion is to be made. Every rule must have a source field, but the other fields may be empty. For example:

Simple Rule

preceding context	source	succeeding context	result
	c	i	s

Notice above, a *c* is turned into an *s*, but only if it is followed by an *i*. Variables can be used to have multiple matches for characters in the context. For example:

Rule with Variables

preceding context	source	succeeding context	result
	c	¬	s

Simple Inclusive Variable

Variable	Meaning
¬	eiy

Ac is turned into an *s*, but only if it is followed by an *e*, *i* or *y*. There are also exclusive variables, which match if the text character is *not* present in the variable's contents. An exclusive variable, with empty contents, will match any character. In normal operation, once a string has been replaced, none of the replacement characters are subsequently checked for matches. However, an additional rechecked result field can be specified that will be rechecked for matches and possibly modified. An example is presented below.

Rule with rechecked result

preceding context	source	succeeding context	result	rechecked result
	k		Σ	• †
	†aa		fi	

In this case, the sequence *kaa* will be converted into Σfi. In the Indic languages, this can be used to capture the interactions between consonants and vowels in a general way.

In the text, rules are written in the following format. The three matching fields are separated from the two resulting fields by an arrow, with the contexts and rechecked result distinguished by strike-through from the adjacent fields.

Example: ~~xyz~~ Ô ~~ab~~;

Example: ~~xyz~~ Ô ~~ab~~;

A Transliteration is designed to hold two sets of rules, so that it can provide transliteration in both directions, such as from Katakana to Latin, and back.

Definitions

The following definitions are used in describing the internal matching process used in a preferred embodiment. There are two different ways to apply Transliteration: one is to a range of text, and the other is on type-in. The definitions apply to both cases.

A rule r_j is made of preceding context, source, succeeding context, result and rechecked result. The preceding context, source, succeeding context are referred to collectively as the source fields, and the result and rechecked result are referred to collectively as the result fields. In the following the length of each part is abbreviated by length(pc), length(s), etc.

A character c_k in the text matches a character c_r in the rule if and only if either

c_r is not a variable and $c_k = c_r$ (this is strong, bitwise identity).

c_r is an *inclusive* variable and c_k e contents $[c_r,]$

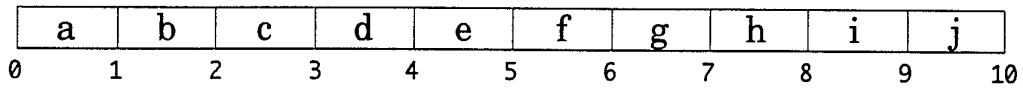
c_r is an *exclusive* variable and $\neg [c_k \text{ e } c_r]$,

A rule **matches** the text within a range at offset *i* if and only if all of the following:

the length(pc) characters before *i* match the preceding context,

the length(s) characters after *i* match the source,

the length(sc) characters, from *i* + length(s) on, match the succeeding context.



The text set forth above is used in the following examples.

Example: bcd Ô z matches at offset 1; and

bcd Ô z matches at offset 2.

A rule **matches** the text within a range at offset *i* **up to** offset *j* if and only if you could add (zero or more) characters after *j* that would cause the rule to match at *i*.

Example: bcde Ô z matches at offset 2 up to offset 4.

A rule spans offset i in the text if and only if there is some j < i such that the rule matches the text at j up to i.

Example: abcdef Ô gh matches at offset 2, and spans offsets 1 through 5.

Example: defx Ô i spans offsets 4 and 5.

(The text characters def match the initial part of the rule, so the offsets between those characters are spanned, even though the x doesn't match.)

Basic Operation on Ranges

With a range of text, only the characters within the range are matched or replaced; the characters outside of the match are completely ignored. The Transliteration operation proceeds as follows. Iterate through the offsets in the range one by one. For each offset *i*, check through the list of rules in the transliteration for the matching rules at that offset. If there are no matching rules, then continue iterating. If there is more than one matching rule, then pick the best match as follows:

Figure 2 is a flowchart setting forth the detailed logic of transliterating text between offsets start and finish in accordance with a preferred embodiment of the invention. Processing commences at function block 200 where index i is initialized to point to the start of the text that is to be transliterated. A test is performed at decision block 210 to determine if the index i has surpassed the finish of the text. If all the text has been processed, then processing is terminated at terminal 220. If the index i is greater than or equal to finish at decision block 210, then currentrule is zeroed, the index j is equated with the first rule found using text(i) as an index, and jmax is equated with the last rule found using text(i) as an index as shown in function block 230.

At decision block 240, a test is performed to determine if the index j has exceeded jmax. If so, then a test is performed at decision block 260 to determine if currentrule is equal to zero. If so, then at function block 270, characters are replaced as indicated and indexes are reset before passing control to decision block 240. If the index j has not exceeded jmax at decision block 240, then Rule(j) is compared to the character located at index i. If they match, then another test is performed at decision block 244 to determine if Rule(j) is better than currentRule. If so, then currentRule is equated to Rule(j) at function block 250, j is incremented at function block 246,

and control is passed to decision block 240. If there is not a match at decision block 242, then j is incremented at function block 240 and control is passed to decision block 240.

In the processing described above, a match *x* is strictly better than a match *y* if and only if either:

the preceding context of *x* is longer than the preceding context of *y*, and the source + succeeding context of *x* is at least as long as the source + succeeding context of *y*

the preceding context of x is at least as long as the preceding context of *y*, and the source + succeeding context of *x* is longer than the source + succeeding context of *y*

Of the matching rules, eliminate all those where there is another matching rule which is strictly better. Of the remainder, pick the first one. (Note that rules are inserted into a Transliteration in order, and that order may be significant).

Example: abc Ô p is better than ab Ô q; yab Ô r is also better than ab Ô q; however, neither abc Ô p nor yab Ô r is better than the other.

In order to speed up matching of rules, the collection of rules is indexed by indexed by the first character in the source of each rule. This does not affect the ordering of the rules, since any two rules that do not share the same first character of the source will never match at the same time, and thus never conflict. Rules with variables in the first position are resolved at the time that they are added to a Transliteration: that is, if the variable has n characters in its contents, then n different rules with the different first letters are added to the Transliteration. When processing forward (either ranges or type-in), the rules are looked up by this first character, then sequentially accessed.

Once a matching rule is identified, a replacement is performed. The source is replaced from the matching rule by the result fields (result + rechecked results). This may change the length of the text, since it may be different in length than the source. Resume iterating, starting at the offset *i* + length(result). This will mean that the rechecked results may be matched and modified. (The plain result can match against the preceding context of another rule, but will not be matched against the source, and thus cannot be subsequently modified).

Operating on Type-In

When transliteration is applied to type-in, the operation is somewhat different. The goal is to produce the same results as would have occurred had the user typed in all of the text without transliteration, then converted it with the range conversion above. In addition, text is converted as it is entered.

It is difficult to predict what characters will follow after an initial text entry, so the process of transliteration cannot be completed until the user has entered additional characters. Example: suppose that there are rules *ph* Ô j and *p* Ô p. If the user has just typed in a *p*, it is impossible to finalize a match because an ambiguity between the two rules exists. When the user types a new character, that character may be modified, *and* preceding characters may also change, since a unique rule may be specified.

The other complication is that it is impossible to predict the starting point of the range, because the user may have just changed to the current transliteration, or just clicked in a new location. So, the text is only converted that could not have been converted without additional characters. Also, if the user is inserting text, all characters after the insertion point are always ignored, so the operation always behaves as if it is at the end of the text.

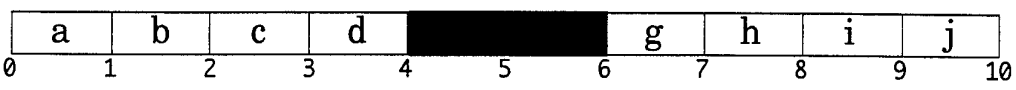
Figure 3 is a flowchart setting forth the detailed logic of transliteration processing between two portions of a text delimited by textStart and textFinished in accordance with a preferred embodiment of the invention. Processing commences at function block 300 where an index i is initialized, and imin is initialized. Then, at decision block 310, a test is performed to determine if i < imin. If not, then i+1 is returned at terminal block 320. If i is less than imin, then at function block 330, the currentrule variable is set equal to zero, j is equated to the rule found using i as an index into text for finding the first rule, and jmax is equated to the rule found using i as an index into text for finding the lastrule. Then, at decision block 340, a test is performed to determine if j is greater than jmax. If so, then i+1 is returned at terminal block 320. If not, then a test is performed at decision block 350 to determine if rule(j) matches from i up to start after counting characters. If so, then i is incremented and processing passes to decision block 310. If not, then, j is incremented and control is passed to decision block 340 for further processing.

The operation proceeds as follows. Suppose that a user types a key which causes *n* characters to be inserted at offset *i*. After inserting the characters at offset *i*, check through all of the rules that span *i*. Find the smallest offset *s* that corresponds

to the start of the source of one of these spanning rules. (Indexing by first character of source does not help in this case, since a backward search is performed, and the current character may be any character in the source or succeeding context of one of these rules. To handle this, the transliteration stores a number *maximumBackup*, which is the length of the longest (source + succeeding context – 1). This is the furthest point back that a spanning rule could start at. The transliteration then searches from *i - maximumBackup* forward to find the first rule that spans the desired index, which determines the index *s*.)

Figure 4 is a flowchart setting forth the detailed logic of type-in transliteration in accordance with a preferred embodiment of the invention. Processing commences at function block 400 where the type-in starting point is determined. Then, at decision block 410, a test is performed to determine if the index i is greater than or equal to finish. If not, then processing is completed at terminal 412. If so, then at function block 414, currentrule is equated to zero, the index j is equated to the first rule indexed by the index i, and jmax is set equal to the last rule indexed by the index i. Then, another test is performed at decision block 416 to determine if j is greater than jmax. If not, then a test is performed at decision block 414 to determine if Rule(j) spans start. If so, then processing is completed, and control is passed to terminal 412. If so, then another test is performed at decision block 430 to determine if currentrule is equal to zero. If so, then the index i is incremented and control is passed to decision block 410. If not, then at function block 440, characters are replaced in accordance with the current rule, and processing is passed to decision block 410.

Once the smallest offset is identified, all the rules that match must be identified at *s* up to $i + n$. If any of these rules also span $i + n$, then no conversions are performed. Otherwise, identify the best match within the transliteration ranges, and perform the substitution. Then, reset *s* according to the result and repeat this processing until done.



In all the following examples, *ef* has been inserted at offset 4 in the above text.

Example: with the rules (cd Ô x; de Ô y), *s* = 3; so convert the *de* to *y*, and do not convert the *cd*·

Example: with the rules (cdm Ô x; cd Ô y), $s = 2$; so convert the *cd* to *y*, and do not affect the *e*.

Example: with the rules (abcdem Ô x; ab Ô y; bc Ô z; bcm Ô w; de Ô v), $s = 1$; so convert the *bc* to *z*, and the *de* to *v*.

Human Interface

An example of the user interface display used to create transliteration rules is presented in Figure 5. This Figure shows a transliteration display in the process of being created in accordance with a preferred embodiment of the invention. The rules are in the top box, with context variables in the center, and test samples at the bottom.

New rules can be added at the end, or inserted in the proper order. If rules do not conflict, or if any rule is inserted after a "better" rule, then it can be automatically reordered. For example, if the rule a Ô x exists, and the user adds the rule ab Ô y, then the inserted rule is reordered before the existing rule since otherwise it would have no effect. New context variables can be added at the end, because order is unimportant. The test source is transliterated into the result whenever any change is made, so that the user can check the correctness of the actions.

Figure 6 is an illustration of a transliteration operation as it would appear on a user's display in accordance with a preferred embodiment of the invention. At label 610, a sentence is typed in with transliteration disabled. Label 680 shows the sentence appearing at label 610 after it has been selected and fully transliterated. Labels 620 to 670 show what the user sees as she types in successive characters from label 610 with input transliteration enabled. At label 620, the user has entered the character "c". At label 630, the user has typed in "ch". At label 640, the user has typed "cho", and there is enough context to invoke the appropriate rule and transliterate the text input into "ちょ". A similar operation is performed on the additional examples appearing at labels 650 to 680.

TRANSLITERATOR CLASS AND METHOD DESCRIPTION

Class TTransliterator

TTransliterator is an abstract base class that transforms text based upon some well-defined rules or algorithm. It descends from TTextModifier, and can be used in conjunction with a word processing engine for inline transliteration during typing. Subclasses of TTransliterator can be used to perform intra-script transformation such as accent composition and upper or lower casing, or inter-script phonetic transcription. Inter-script transliteration provides an alternative way of entering non-Roman text. For example, instead of using a Hebrew keyboard, Hebrew text can be entered with an American keyboard using Roman to Hebrew transliteration. This class is also used to aid typing by providing smart quotes and other punctuation modifiers. TTransliterator provides methods for translating text as well as reversing the effect. These methods must be overridden by the concrete subclasses. TTransliterator is designed to enable chaining multiple objects together. One or more such objects can work within the editable text classes to allow for transliteration during typing.

The replacement text will also have the correct styles. E.g., if the source text is "*ae***iou**", and the replacements are "Æ" for "a", "I" for "i", "O" for "o" and "U" for "u", then the replacement text will be: "*Æ***IOU**". If a replacement for some characters is a larger number of characters, then the last style will be extended. For example, if the source is "**W**hy", and the replacements are "VV for W", then the replacement text is: "**VV**hy". The "C" source code used to implement a preferred embodiment in accordance with the description above is set forth below.

Public Methods

```
/*************************************************************************
 TTextModifier overrides
 *************************************************************************/
virtual Boolean WantEvent(const TEvent& event) const=0;
    /* Override to always return FALSE.*/

virtual Boolean ProcessEvent(const TEvent& event,
                             const TModelCommand& currentCommand)=0;
    /* Override to always return FALSE.*/
```

```
virtual Boolean ProcessNewText(const TBaseText& newText,
                               const unsigned long numDeletes,
                               const TInsertionPoint& insertionPoint,
                               TTexTTextRange& rangeToDelete,
                               TBaseText& textToInsert)=0;
    /* Override to call Translate(). */

virtual void ProcessNewSelection(const TInsertionPoint& insertionPoint,
                                 unsigned long length)=0;
    /* Override to do nothing. Transliterators don't care about new selection */

/**************************************************************************
*
 These methods will directly translate(back the TBaseText argument.

 Subclass: This method calls the appropriate Translate() method and
           should not be overridden.
**************************************************************************
/
          Boolean SimpleTranslate(TBaseText& text) const;
     Boolean SimpleTranslateBack(TBaseText& text) const;

/**************************************************************************
*
 Translate() takes a sourceText, and a range within that text, and either modifies the
text directly, or produces the replacement text and the range of characters to replace.
In the latter case, the replacement range is always a subset of the source range. It is
up to the caller to call the appropriate text methods to actually substitute the
replacement. The actual transformation is defined by a concrete subclass.
**************************************************************************
/
virtual Boolean Translate (const TBaseText& sourceText,
                           const TTextRange& sourceRange,
                           TBaseText& replacementText,
                           TTexTTextRange&
replacemenTTextRange) const = 0;

virtual Boolean Translate (TBaseText& sourceText,
```

```
                                                const TTextRange& sourceRange,
                                                TTextRange&
    replacemenTTextRange,
                                                unsigned long& numNewChars)
    const = 0;
        /* This method is similar to the one above, the difference being that given a
          text object and a range within it, this method will directly change the
              text object. */

    /**************************************************************************
    *
    TranslateBack() takes the same arguments, but translates in the reverse direction,
    from target characters back to source characters using a translation mechanism
    defined  by a concrete subclass. Note that these methods are not suitable for use with
    keyboard entry
    **************************************************************************
    /
    virtual Boolean TranslateBack (const TBaseText& sourceText,
                                                const TTextRange&
    sourceRange,
                                                TBaseText& replacementText,
                                                TTextRange&
    replacemenTTextRange) const = 0;

    virtual Boolean TranslateBack (TBaseText& sourceText,
                                                const TTextRange&
    sourceRange,
                                                TTextRange&
    replacemenTTextRange,
                                                unsigned long&
    numNewChars) const = 0;
```

Class TRuleBasedTransliterator

TRuleBasedTransliterator is derived from TTransliterator. It uses a set of context- sensitive rules to transform text, and a parallel set to reverse this action. These rules are designed such that a knowledgeable non-programmer can edit them

for localization. Roman rule-based transliteration is available for Japanese (kana), Hebrew, Arabic, Greek, Russian, and Devanagari.

TRuleBasedTransliterator also has the capability of specifying a range variable. These variables can then be used in the rules to provide some simple pattern-matching functions. An example of the foregoing is presented below.

Public Methods

```
/**************************************************************************
Constructor
**************************************************************************/
TRuleBasedTransliterator(const TFile& rulesFile);
    /* Instantiate a transliterator based on the rules in a file. */

/**************************************************************************
*
TTransliterator overrides. Translate() uses a set of context sensitive rules to
perform the translation. See TTransliterateRule for more details. Rules are applied
according to the principles of Partial and Multiple Replacements, as in the
discussion above
**************************************************************************
/
virtual Boolean Translate (const TBaseText& sourceText,
                           const TTextRange& sourceRange,
                           TBaseText& replacementText,
                           TTexTTextRange&
replacemenTTextRange) const;

virtual Boolean Translate (TBaseText& sourceText,
                           const TTextRange& sourceRange,
                           TTextRange&
replacemenTTextRange,
                           unsigned long& numNewChars)
const;

/**************************************************************************
*
```

TranslateBack() uses a set of context sensitive rules to performa the transliteration. takes the same arguments, but translates in the reverse direction, from target characters back to source characters. For example, with a Roman to Hebrew transliterator, Translate will go from Roman to Hebrew; TransliterateBack will go from Hebrew to Roman

```
**********
/
virtual Boolean TranslateBack (const TBaseText& sourceText,
                               const TTextRange&
sourceRange,
                               TBaseText& replacementText,
                               TTextRange&
replacemenTTextRange) const = 0;

virtual Boolean TranslateBack (TBaseText& sourceText,
                               const TTextRange&
sourceRange,
                               TTextRange&
replacemenTTextRange,
                               unsigned long&
numNewChars) const = 0;

/**********
*
 Add and remove rules.
**********
/
virtual void AddTranslationRule (const TTransliterateRule&    rule);
virtual void AddTranslationBackRule(const TTransliterateRule&      rule);
virtual void RemoveTranslationRule (const TTransliterateRule&      rule);
virtual void RemoveTranslationBackRule(const TTransliterateRule& rule);

/**********
*
```

```
Methods for range variables. These variables an be used for a limited degree of
"wildcarding". Any character that is not in the source or target set can be designated
to be a range variable. Effectively, that character will match against any of the
characters in the specified range. Inverse ranges can also be specified: in that case, a
character matches against any character not in the specified range.
For example, if $ is defined as a range variable equal to "ei", then the two rules:
        c[$>S
        c>K
will cause "c" in front of "i" or "e" to be converted to "S", and in front of anything
else to be converted to "K".
**********************************************************************
/
virtual void AddVariable (TransliterateVariable variableName,
                                                const TBaseText& variableValue,
                                                Boolean inverse = FALSE);

virtual Boolean IsVariable (TransliterateVariable    ch) const;
        /* Checks to see if a particular unicode character represents a variable. */

/**********************************************************************
*
Create iterators for the rules.
**********************************************************************
/
virtual TIterator* CreateIterator () const;
virtual TIterator* CreateTranslateBackIterator () const;
```

Class TTransliterateRule

TTransliterateRule implements a context sensitive transliteration rule. It is used within a TRuleBasedTransliterator.

```
Public Methods
/**********************************************************************
*
Construct a rule with four components. Note that these components cannot
exceed 256 charcters in length (This is an arbitrary limitation imposed by
the desire to save as much space in rule storage as possible).
```

```
**********************************************************************
/
TTransliterateRule(const TBaseText&              keyText,
                   const TBaseText&                    resultText,
                   const TBaseText* const  preceedContext=NIL,
                   const TBaseText* const  succeedContext=NIL);

/**********************************************************************
*
Use the rule described by the current object to perform translation. This
method is called by TRuleBasedTransliterator::Translate()

**********************************************************************
/
virtual long Translate(TBaseText& sourceText, unsigned long&sourceOffset,
unsigned long&numReplacedChars, unsigned long&numNewChars) const;
/**********************************************************************
*
Determines if the rule defined by this object applies to a given text object

**********************************************************************
/

  virtual Boolean DoesRuleApply(const TBaseText&text, const unsigned long
  textOffset, const TDictionary&variableTable, Boolean&variableMatch,  unsigned
  long&translatableChars) const;
```

Class TKoreanTransliterator

TKoreanTransliterator implements Korean Jamo<->Hangul transliteration. It is a subclass of TTransliterator, however, the algorithm used is completely different than the rule-based transliteration implemented by TRuleBasedTransliterator. Background: Each Hangul character consists of two to three components, called Jamo's. The first and second components are mandatory; however, the third is optional. The transliterator assumes that all defined Jamo and Hangul characters can be in the source text, even though some complex Jamos cannot be entered from certain keyboards. This means that another preprocessors is needed to compose the complex Jamos from the "type-

able" Jamo. The reason that this step is separated from Jamo-Hangul transliteration is that the Jamo-Jamo composition is highly dependent on the keyboard layout where the Jamo-Hangul process is not.

Class THexTransliterator

THexTransliterator is derived from TTransliterator. It transforms hex numbers between one and four digits to their unicode representations. When used inline, THexTransliterator provides a simple input method for generating Unicode characters that cannot be entered with a keyboard.

Class TSystemTransliterator

TSystemTransliterator is the user's interface to the system's transliterators. It can be used to get and set the current transliteration, and to query for the available transliterators on the system as well as perform all the public protocol of TTranslitator. It is notified when a change occurs in the system transliterator.

Public Methods
(TTransliterator overrides)

```
/**************************************************************************
The following constants define the transliterators currently available
in the system
**************************************************************************/
static const TToken& kRomanTransliterator;
static const TToken& kJapaneseTransliterator;
static const TToken& kKatakanaTransliterator;
static const TToken& kArabicTransliterator;
static const TToken& kCapitalizeTransliterator;
static const TToken& kHebrewTransliterator;
static const TToken& kDevanagariTransliterator;
static const TToken& kRussianTransliterator;
static const TToken& kSymbolTransliterator;
static const TToken& kGreekTransliterator;

/**************************************************************************
*
Query for all available transliterators. Returns a TCollection of TTokens,
```

```
    each being the name of a transliterator.
    ********************************************************************************
    /
    virtual void AvailableTransliterators(TCollection&        transliterators) const;

    /********************************************************************************
    *
    Gets and sets the system transliterator by name. If SetSystemTransliterator()
    is called, the next call to any public method in this class will reflect the change.
    ********************************************************************************
    /
    virtual void GetSystemTransliteratorName(TLocaleName& name) const;
    virtual void SetSystemTransliterator(const TLocaleName& name);
```

EXAMPLES IN ACCORDANCE WITH A PREFERRED EMBODIMENT

Creating a TRuleBasedTransliterator from a Text File

The TRuleBasedTransliterator object can be created from a text file. The file contains an ordered sequence of definitions, range variables, forward rules and backward rules. The following example consists of fragments of a text file specification for Devanagari (Hindi). Comments are preceded with a hash mark (#). The following are character identifier definitions. ($XXXX indicates a Unicode character.) Once an identifier definition has been processed, then any occurrence of that identifier within brackets will be replaced by the right-hand side. In these text file examples, the x) preceding context, y) source, z) succeeding context, a) result, b) rechecked result are represented as: x] y [z > a | b as discussed in more detail below. For example:

```
ka=$915
kha=$916
ga=$917
gha=$918
nga=$919
virama=$94D
aa=$93E
```

```
i=$93F
ii=$940
u=$941
uu=$942
rh=$943
lh=$944
e=$947
ai=$948
o=$94B
au=$94C
The following are range variables:
&:{virama}{aa}{ai}{au}{ii}{i}{uu}{u}{rrh}{rh}{lh}{e}{o}
∂:bcdfghjklmnpqrstvwxyz
The following convert from Latin letters to Devanagari letters.
kh>{kha} | {virama}
k>{ka} | {virama}
q>{ka} | {virama}
{virama}aa>{aa}
{virama}ai>{ai}
{virama}au>{au}
{virama}ii>{ii}
{virama}i>{i}
# otherwise convert independent forms when separated by ': k'ai ->
{ka}{virama}{wai}
{virama}'aa>{waa}
{virama}'ai>{wai}
{virama}'au>{wau}
{virama}'ii>{wii}
# convert to independent forms at start of word or syllable:
# e.g. keai -> {ka}{e}{wai}; k'ai -> {ka}{wai}; (ai) -> ({wai})
aa>{waa}
ai>{wai}
au>{wau}
ii>{wii}
i>{wi}
```

The following rules convert back from Devanagari letters to Latin letters. Note that a less-than sign is used for the backwards rules.

```
# normal consonants
{kha}[&<kh
{kha}<kha
{ka}{virama}[{ha}<k'
{ka}[&<k
{ka}<ka
{gha}[&<gh
{gha}<gha
# dependent vowels (should never occur except following consonants)
{aa}<aa
{ai}<ai
{au}<au
{ii}<ii
# independent vowels (when following consonants)
∂]{waa}<'aa
∂]{wai}<'ai
∂]{wau}<'au
∂]{wii}<'ii
```

Translating Text Using the System Transliterator

```
void TranslateText(TBaseText& text)
{
        // Instantiates a TSystemTransliterator. Will perform transliteration
        // based on the object currently chosen for the system. The default
        // is "Roman", which includes accent composition and smart quotes.
        TSystemTransltierator transliterator;
        TText replacementText;
        TTextRange replacemenTTextRange;
        if (transliterator.Translate(text, TTextRange(0, text.Length()),
                                                        replacementText,
replacemenTTextRange))
        {
                text.DeleteText(replacemenTTextRange.RangeBegin(),

replacemenTTextRange.RangeLength());
                text.InsertText(replacementText, text.Length());
        }
}
```

Creating and Using Pre-Defined Transliterator

```
void UpperCase(TBaseText& text)
{
        // Instantiates the "Capitalize" transliterator. Note that
        // creating a transliterator is very expensive since it has
        // to either build or stream in the table. Therefore it's
        // best to create once only.
        static TRuleBasedTransliterator
upperCase(TSystemTransliterator::kCapitalize);      TText replacementText;
        TTextRange replacemenTTextRange;
        if (upperCase.Translate(text, TTextRange(0, text.Length()),
         replacementText, replacemenTTextRange))
        {
                text.DeleteText(replacemenTTextRange.RangeBegin(),
        replacemenTTextRange.RangeLength());
                text.InsertText(replacementText, text.Length());}
        // to change text into lower case, call the TranslateBack()
        // method.
void ChangeSystemTransliterator(const TLocaleName& newTransliteratorName)
{
        // Instantiates a TSystemTransliterator.
        TSystemTransliterator transliterator;
        transliterator.SetCurrentTransliterator(newTransliteratorName);
}
```

While the invention has been described in terms of a preferred embodiment in a specific system environment, those skilled in the art recognize that the invention can be practiced, with modification, in other and different hardware and software environments within the spirit and scope of the appended claims.

CLAIMS

Having thus described our invention, what we claim as new, and desire to secure by Letters Patent is:

1. A computer system for transliterating text as it is input to the computer, comprising:
   (a) a storage for storing a plurality of transliteration rules;
   (b) a display for displaying text as it is entered into a computer; and
   (c) means for selecting a transliteration rule from the plurality of transliteration rules and applying the rule to the text and displaying the resultant text on the display.

2. The system as recited in claim 1, including processing means for initiating the text processing operation through an iconic operation.

3. The system as recited in claim 2, including processing means for initiating the text operation by double-clicking on an icon.

4. The system as recited in claim 2, including processing means for drop-launching the text operation.

5. The system as recited in claim 1, including:
(a) a storage for storing at least one preceding context rule;
(b) a storage for storing a source; and
(c) a storage for storing at least one succeeding context rule.

6. The system as recited in claim 5, including a processor for applying the rules to text and marking appropriate text for rechecking.

7. The system as recited in claim 5, including means for applying inclusive and exclusive variables.

8. The system as recited in claim 5, including hashing means for increasing the performance of rule lookup in the storage.

9. The system as recited in claim 4, including means for calculating an optimal number of characters to process before applying the means for transliterating.

10. The system as recited in claim 1, including a display for composing, modifying and storing rules and variables for use in the transliteration operation.

11. The system as recited in claim 4, including means for transcribing from one script to another during input operations.

12. The system as recited in claim 1, including means for converting multiple characters into a single character representation of the multiple characters, storing the single character representation in a storage and displaying the single character representation on a display.

13. The system as recited in claim 1, including an object for transliterating forward and backward.

14. A method for transliterating text as it is input to a computer, comprising the steps of:
(a) storing a plurality of transliteration rules;
(b) displaying text as it is entered into the computer; and
(c) selecting a transliteration rule from the plurality of transliteration rules and applying rule to the text and displaying the resultant text on the display.

15. The method as recited in claim 14, including the step of initiating the text processing operation through an iconic operation.

16. The method as recited in claim 15, including the step of initiating the text operation by double-clicking on an icon.

17. The method as recited in claim 15, including the step of drop-launching the text operation.

18. The method as recited in claim 14, including the steps of:
(a) storing at least one preceding context rule;
(b) storing a source; and
(c) storing at least one succeeding context rule.

19. The method as recited in claim 18, including the step of applying the rules to text and marking appropriate text for rechecking.

20. The method as recited in claim 17, including the step of applying inclusive and exclusive variables.

21. The method as recited in claim 17, including the step of increasing the performance of rule lookup in the storage.

22. The method as recited in claim 16, including the step of calculating an optimal number of characters to process before applying the means for transliterating.

23. The method as recited in claim 14, including the step of composing, modifying and storing rules and variables for use in the transliteration operation.

24. The method as recited in claim 16, including the step of transcribing from one script to another during input operations.

25. The method as recited in claim 14, including the step of converting multiple characters into a single character representation of the multiple characters, storing the single character representation in a storage and displaying the single character representation on a display.

26. The method as recited in claim 14, including the step of transliterating forward and backward.

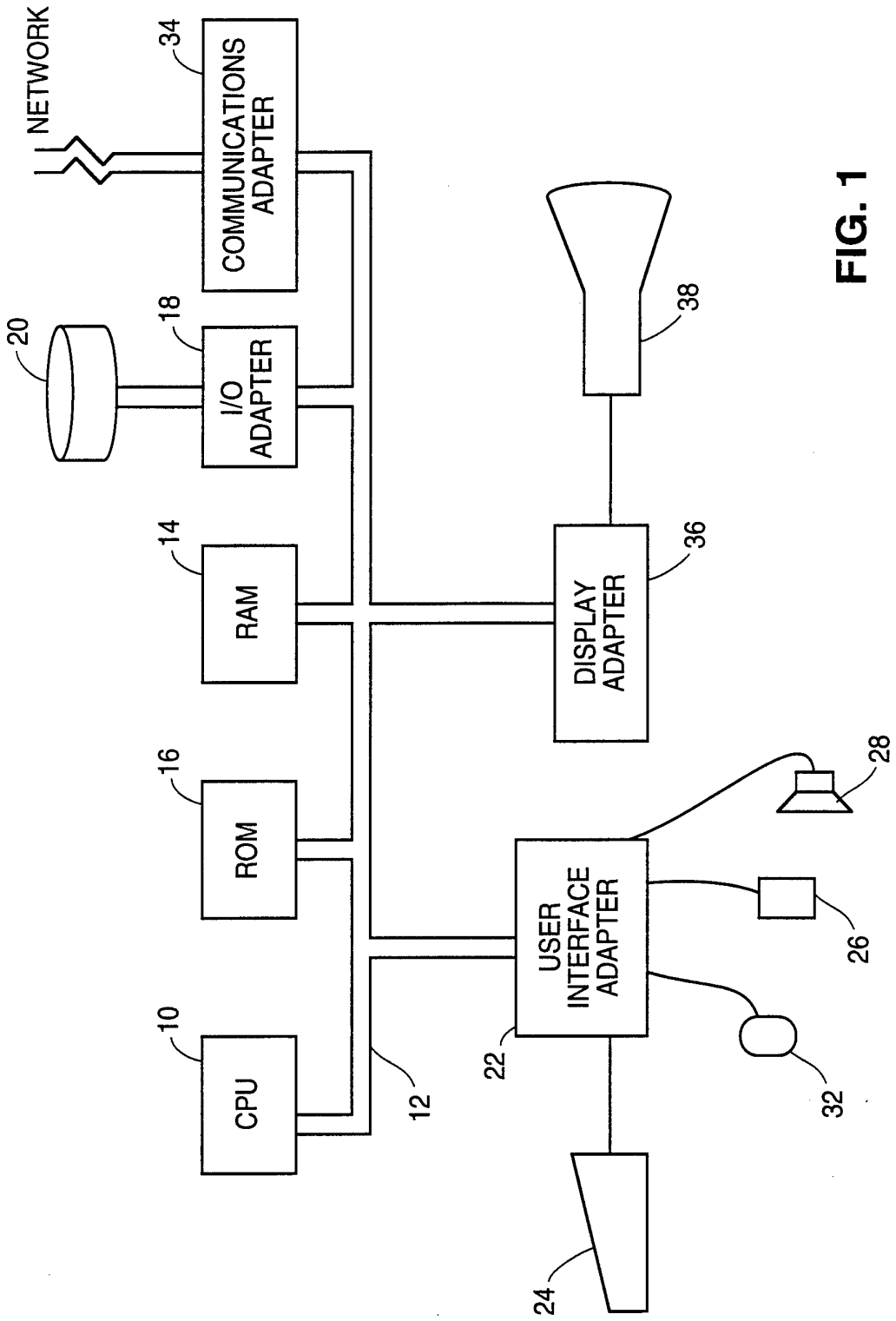

NETWORK
34
COMMUNICATIONS ADAPTER
20
18
I/O ADAPTER
38
14
RAM
36
DISPLAY ADAPTER
16
ROM
28
USER INTERFACE ADAPTER
26
10
CPU
12
22
32
24

FIG. 1

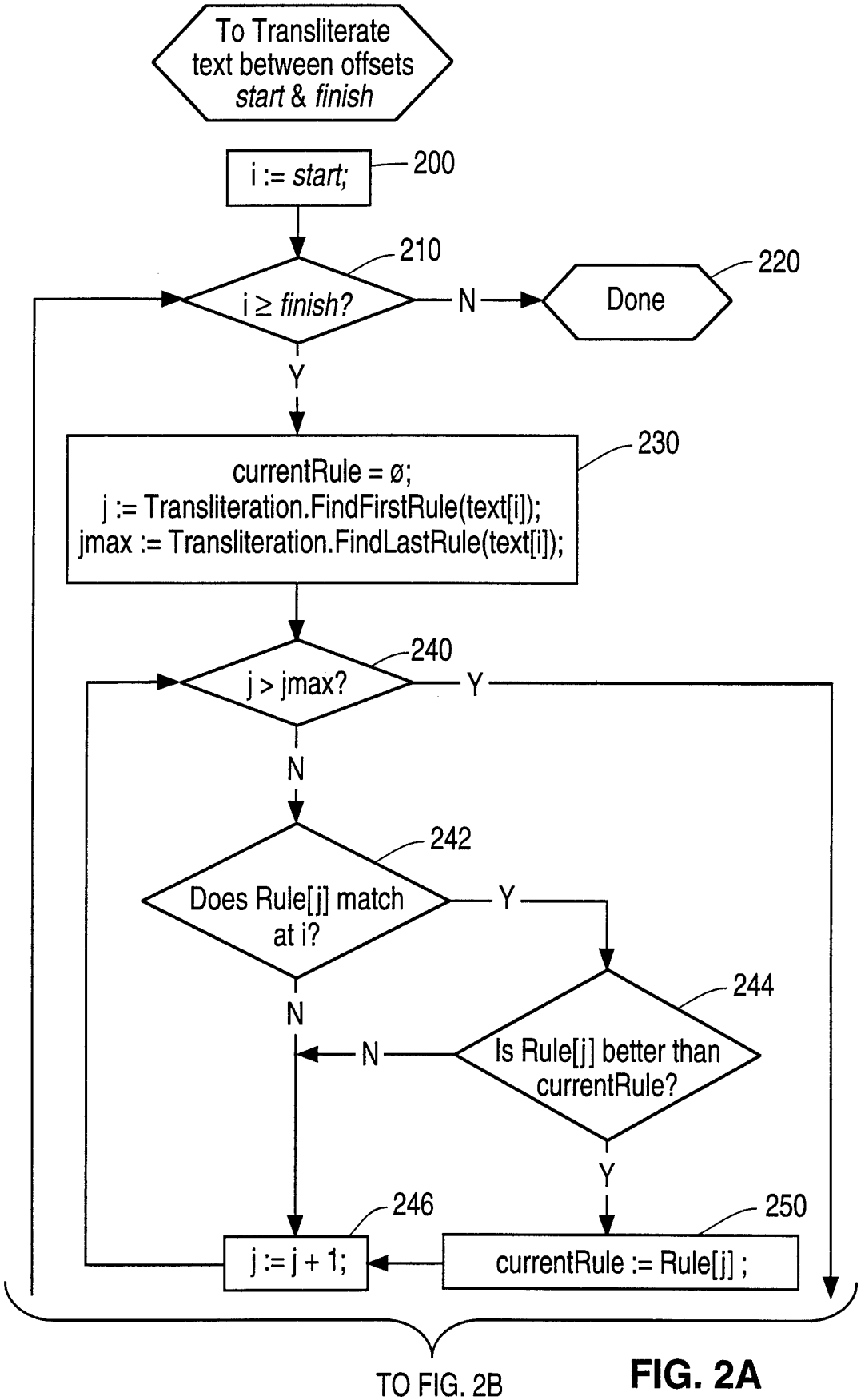

To Transliterate text between offsets start & finish
i := start;
200
210
i ≥ finish?
N
Done
220
Y
currentRule = ø;
j := Transliteration.FindFirstRule(text[i]);
jmax := Transliteration.FindLastRule(text[i]);
230
240
j > jmax?
Y
N
242
Does Rule[j] match at i?
Y
N
244
Is Rule[j] better than currentRule?
N
Y
246
250
j := j + 1;
currentRule := Rule[j] ;
TO FIG. 2B

FIG. 2A

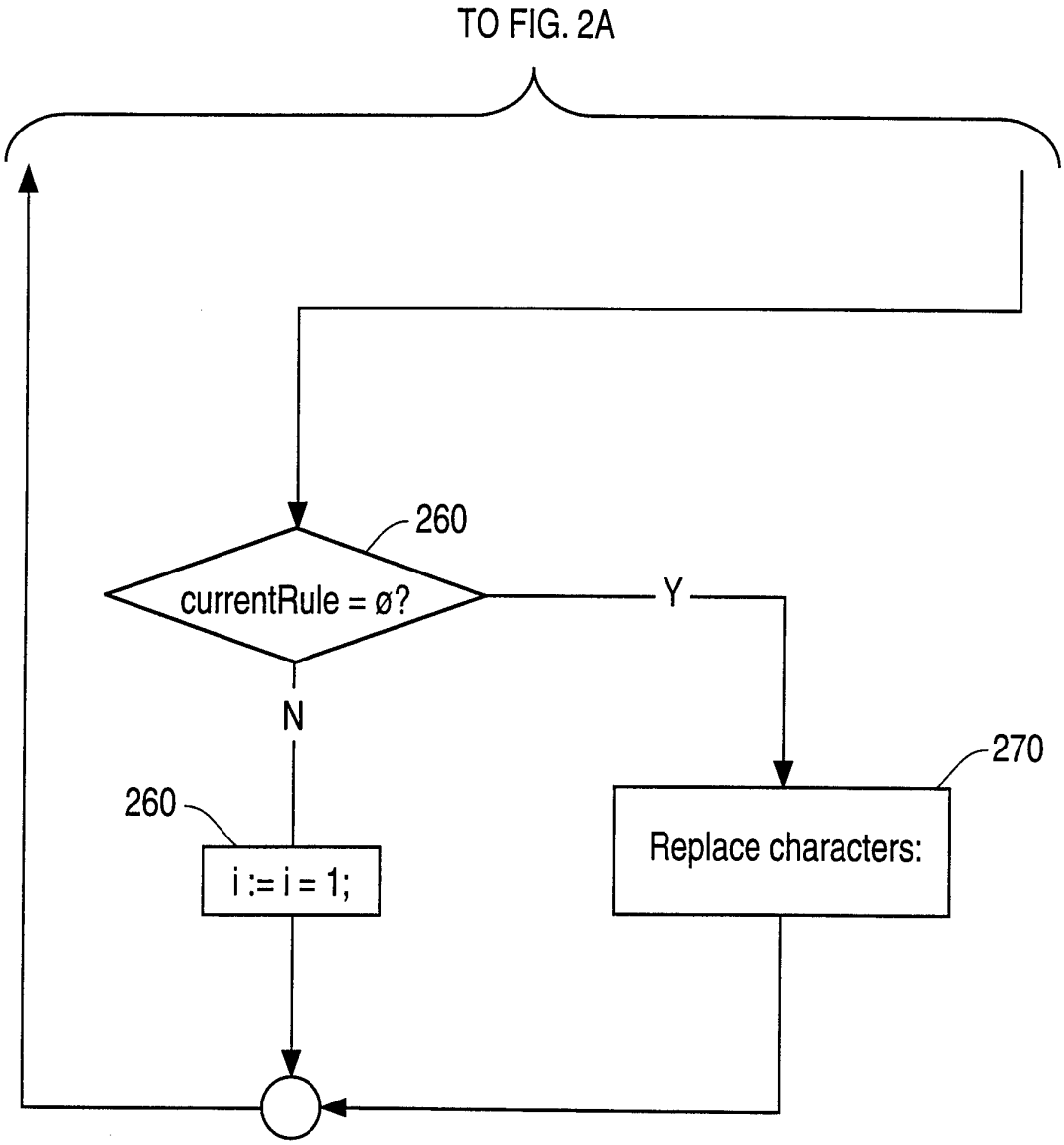

TO FIG. 2A
260
currentRule = ø?
Y
N
270
260
Replace characters:
i := i = 1;

FIG. 2B

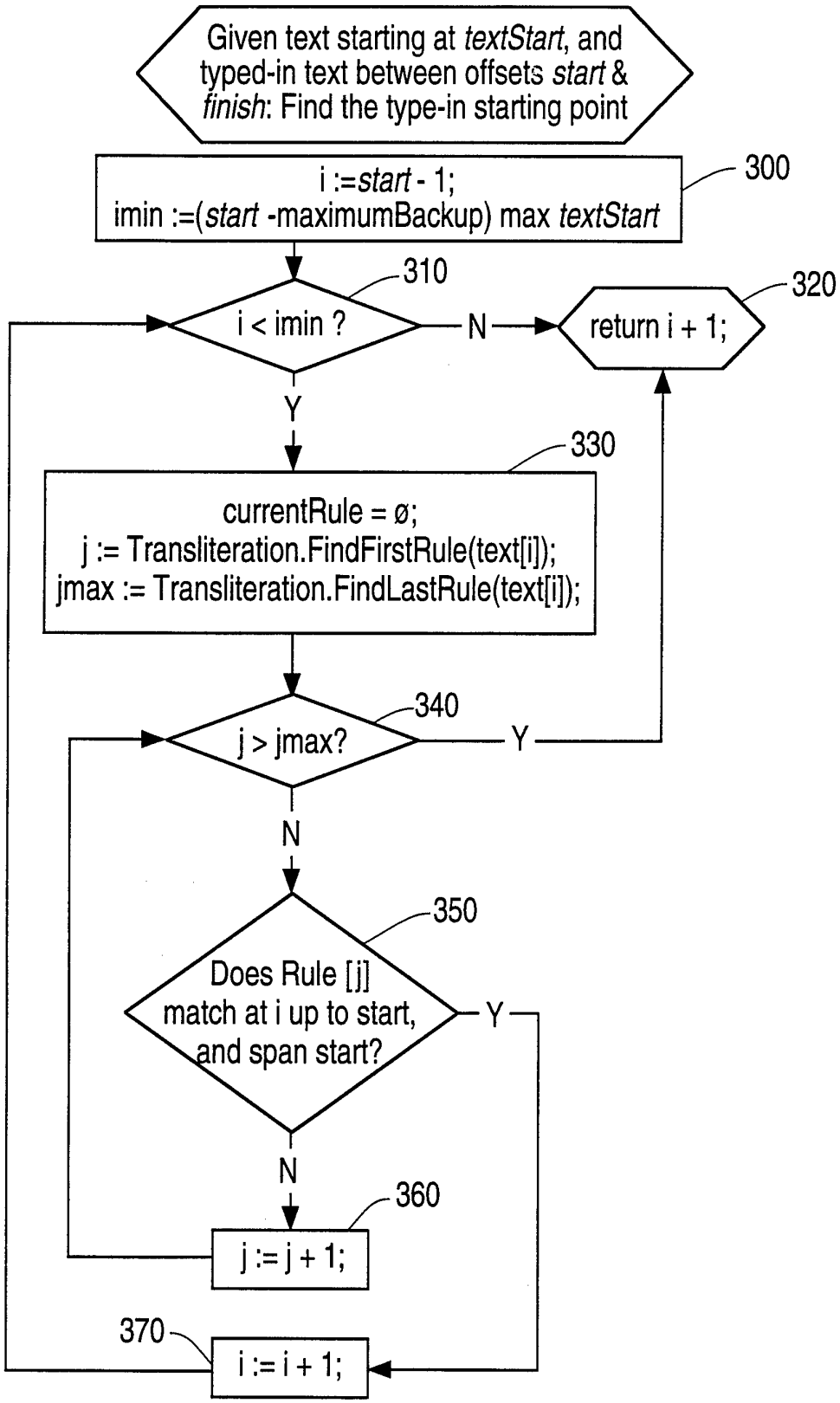

Given text starting at textStart, and typed-in text between offsets start & finish: Find the type-in starting point
i :=start - 1;
imin :=(start -maximumBackup) max textStart
300
310
i < imin ?
N
return i + 1;
320
Y
330
currentRule = ø;
j := Transliteration.FindFirstRule(text[i]);
jmax := Transliteration.FindLastRule(text[i]);
340
j > jmax?
Y
N
350
Does Rule [j] match at i up to start, and span start?
Y
N
360
j := j + 1;
370
i := i + 1;

FIG. 3

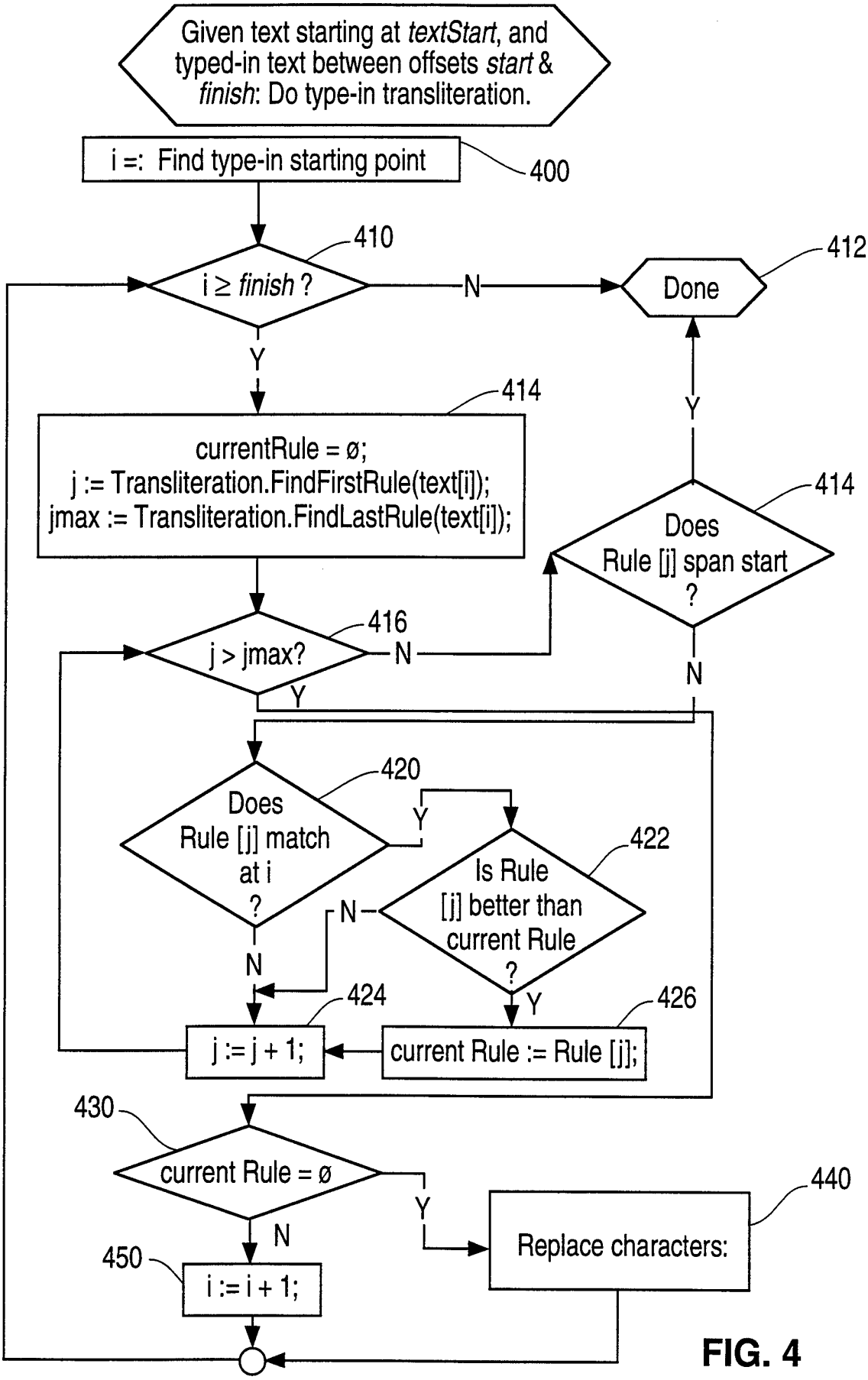

Given text starting at *textStart*, and typed-in text between offsets *start* & *finish*: Do type-in transliteration.
i =: Find type-in starting point
400
410
i ≥ *finish* ?
N
412
Done
Y
414
currentRule = ø;
j := Transliteration.FindFirstRule(text[i]);
jmax := Transliteration.FindLastRule(text[i]);
Y
414
Does Rule [j] span start ?
416
j > jmax?
N
Y
N
420
Does Rule [j] match at i ?
Y
422
Is Rule [j] better than current Rule ?
N
N
Y
424
j := j + 1;
426
current Rule := Rule [j];
430
current Rule = ø
Y
N
440
Replace characters:
450
i := i + 1;

FIG. 4

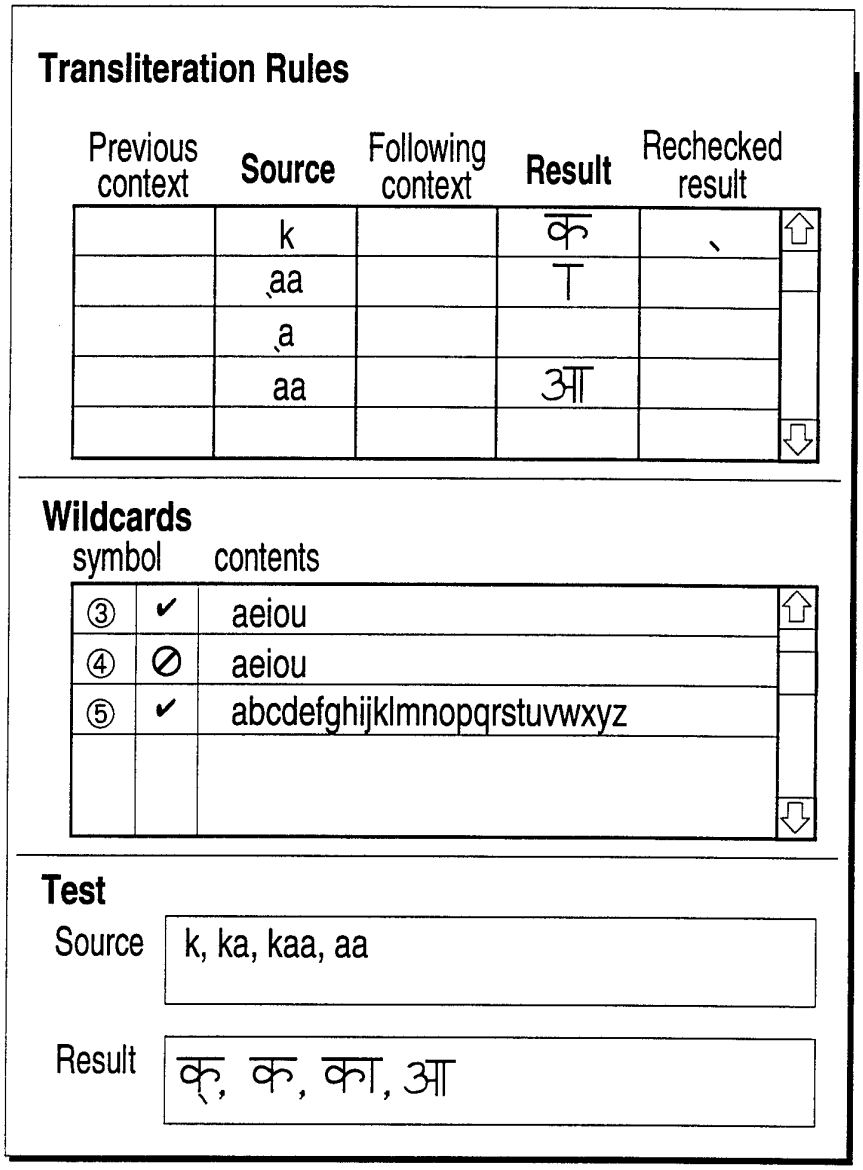

Transliteration Rules
Previous context
Source
Following context
Result
Rechecked result
k
क
aa
T
a
aa
आ
Wildcards
symbol
contents
③ ✔ aeiou
④ ⊘ aeiou
⑤ ✔ abcdefghijklmnopqrstuvwxyz
Test
Source
k, ka, kaa, aa
Result
क्, क, का, आ

FIG. 5

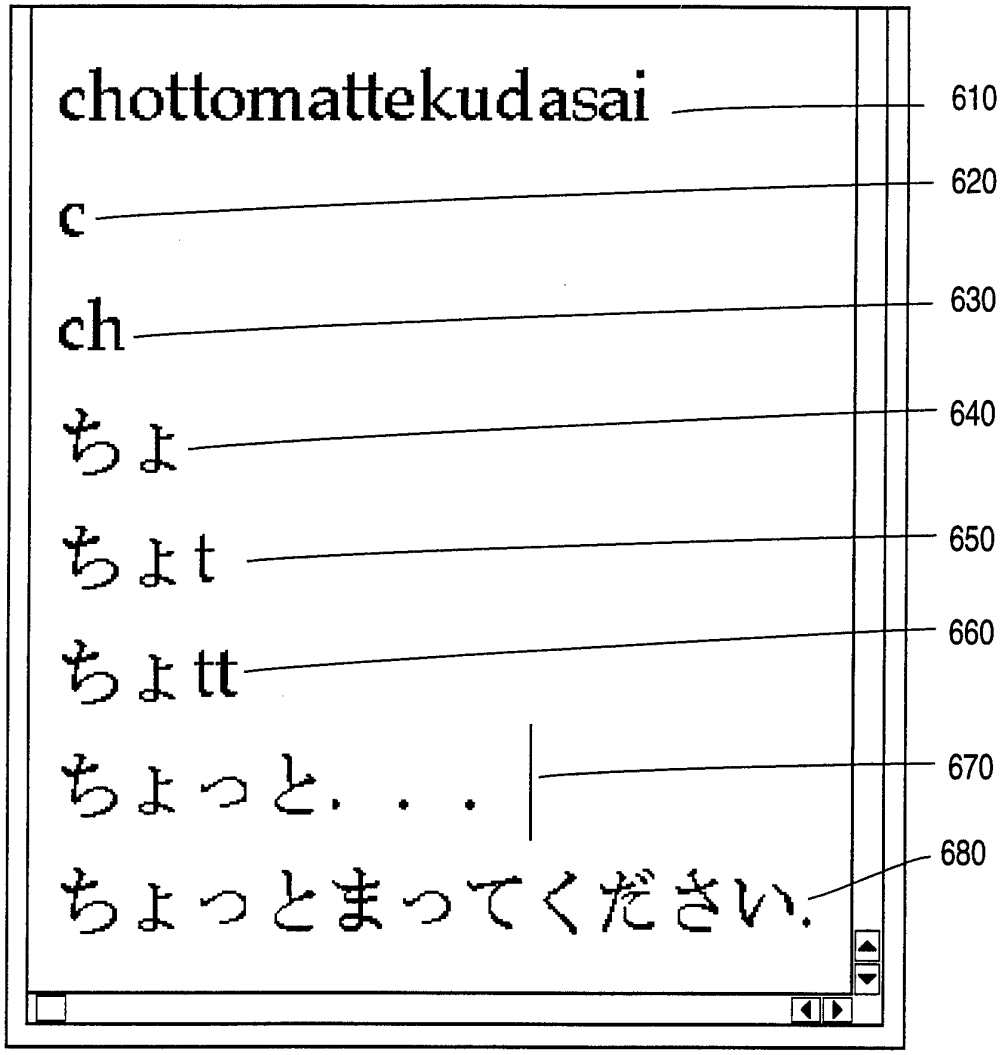

chottomattekudasai
610
c
620
ch
630
ちょ
640
ちょt
650
ちょtt
660
ちょっと. . .
670
ちょっとまってください.
680

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 94/00081

A. CLASSIFICATION OF SUBJECT MATTER
IPC 5 G06F15/20

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 5 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	JOURNAL OF THE INSTITUTION OF ELECTRONICS AND TELECOMMUNICATION ENGINEERS, INDIA, vol. 30, no. 6 , November 1984 pages 243 - 245 SINHA R.M.K., SRINIVASAN B. 'MACHINE TRANSLITERATION FROM ROMAN TO DEVANGARI AND DEVANGARI TO ROMAN'	1,5, 10-12,14
Y		2-4,6, 13, 15-17, 23-26

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
3 May 1994	24. 05. 94.

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Suendermann, R

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 94/00081

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	DATABASE WPI Week 9233, Derwent Publications Ltd., London, GB; AN 92-274839 ANONYMOUS 'Method for implementing action-object direct manipulation' see abstract & RSEARCH DISCLOSURE vol. 339, no. 091 , 20 June 1992 ---	2-4,6, 13, 15-17, 23-26
X	JOURNAL OF INFORMATION PROCESSING, JAPAN vol. 8, no. 4 , 1985 pages 300 - 306 SHIBAYAMA M., HOSHINO S. 'Implementation of an intelligent Thai computer terminal' ---	1,5,12, 14
A	PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON TEXT PROCESSING SYSTEMS 26 October 1984 , DUBLIN, IRELAND pages 133 - 144 GRIFFIN C. 'Typsetting Exotic Languages at Oxford University' ---	1,14
A	GB,A,2 125 197 (PINGYI ZHI ET AL.) 29 February 1984 see the whole document ---	1,14
A	DATABASE WPI Week 9012, Derwent Publications Ltd., London, GB; AN 90-088682 ANONYMOUS 'Automating program processing for windowing system' see abstract & RESEARCH DISCLOSURE vol. 310, no. 049 , 20 February 1990 -----	2-4,6, 15-17

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US 94/00081

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
GB-A-2125197	29-02-84	NONE	