Chicon --- A Chinese Text Manipulation Language

Kam-Fai Wong, Vincent Y. Lum, Wai-Ip Lam

Department of Systems Engineering and Engineering Management
Chinese University, Shatin, N.T., Hong Kong.
(email: {kfwong,vlum,wilam}@se.cuhk.edu.hk)

Summary

Text processing is an important computer application. Due to its importance, a number of text manipulation programming languages have been devised, e.g. Icon. These programming languages are very useful for applications such as natural language processing, text analysis, text editing, document formatting, text generation, ... etc. However, they were mainly designed to handle English texts and are ineffective for Chinese. This is because English and Chinese texts are represented very differently in a computer. An English character is mainly represented in 7-bit ASCII and its Chinese counterpart commonly in 16-bit GB or BIG-5. This difference makes direct application of English-based text manipulation programming languages to Chinese erroneous, e.g. application of Icon to reverse a string of Chinese characters. In this paper, a new dialect of Icon, referred to as Chicon (i.e. Chinese Icon), is proposed. In the design of Chicon, new data types were introduced to differentiate pure English and English/Chinese mixed texts. In addition, existing Icon text manipulation functions were modified to account for Chinese texts. Experiments have shown that Chicon not only could overcome the problems of Chinese processing in Icon, but its execution speed was actually superior to Icon in handling Chinese. Furthermore, application of Chicon to a real sized problem, namely word segmentation, has proved that the language is practical.

Key Words: Icon computer language; compiler; Chinese information processing

1 Introduction

The number of Chinese documents handled daily grows rapidly, e.g., on the Internet. Manual text processing is becoming more and more impractical. There is, therefore, a desperate need for computer text processing technology applicable to Chinese language. The importance of computer text processing has long been appreciated in the Western world. This led to the development of the programming language, SNOBOL [1] and its successor Icon [2,3,4,5] which were designed with efficient built-in text manipulation utilities. These languages are suitable for applications in natural language processing, compiler, text editing and formatting, questionnaire analysis... etc.

Direct application of the aforesaid programming languages to Chinese text is ineffective. This is because Chinese characters are typically represented using a 16-bit codeset, e.g. BIG-5 (大五碼) was designed to represent *fantizi* (繁體字) and GB (國標碼) [6] was designed to represent *Jiantizi* (簡體字). The former is commonly

used in Taiwan and Hong Kong, and the latter in China, Malaysia and Singapore. On the other hand, English characters are represented in 7-bits ASCII. On top of the difference in code size, the most significant bit of the Chinese code (i.e. BIG-5 or GB) is always set to a 1. Consider the following: If an English-oriented text manipulation language was directly applied to Chinese, it would not be able to differentiate Chinese and English characters and would simply treat a Chinese character as two single bytes. This would lead to an erroneous result. To appreciate the problem, let's take a simple example in string searching: the text to be searched consists of 4 characters, namely {0x41,0x42,0xA4,0x41,0x42}- the first 2 bytes are ASCII characters, i.e. 'A' and 'B', the third and the fourth together form a BIG-5 Chinese character, i.e. '乙', and the fifth is again an ASCII character. If the text was compared with the pattern string {0x41,0x42}, i.e. {'A','B'} using an English-oriented language, there would be two hits, i.e. the first/second and fourth/fifth character pairs. However, the correct answer should only be the former pair. This is, in fact, the well-known misalignment problem in Chinese/English string matching [7].

One possible solution to the above problem is that the programmer writes a small procedure to detect Chinese and English characters at run-time, e.g. using the following simple algorithm:

```
1       IF the most significant bit of the input byte is a 1 THEN
2               read the next input byte and concatenate it to the first character;
3               return concatenated double byte string as a Chinese character;
4       ELSE return the single input byte as an English character.
```

This solution is, however, ad hoc and requires the programmer to keep track of every input character. This renders English-oriented text processing languages for Chinese cumbersome. Furthermore, it inevitably incurs a fair amount of run-time overhead and can seriously affect the system performance in large text processing applications, e.g. corpus analyses in Chinese natural language processing.

It is worth noting that ISO10646 [8], the so called Universal Codeset, is the latest international character set standards which include different character set standards worldwide in a fix-length format. For example, it includes Asian languages such as GB2312, BIG-5 and Han characters. Although the misalignment problem disappears with ISO10646, the huge volume of BIG-5 and GB documents existing renders processing of text with BIG-5/GB and English mixture important. This is analogous to the continuous support of COBOL programming in the computing community for thousands of existing business applications were written in this language. Converting all applications from COBOL to more modern programming languages is feasible, but it is impractical.

In this paper, a new Chinese text manipulation programming language, referred to as Chicon, is introduced. Strictly speaking, Chicon is a dialect of Icon. It follows closely the syntax of Icon; and in addition to the existing built-in English-oriented text manipulation functions, Chicon provides a set of similar functions for manipulation of Chinese text. Moreover, many operators and basic data types have been introduced in Chicon for efficient processing of Chinese text. These facilities cannot simply be implemented as a set of functions included in a library. Icon was selected as the underlying language model because of its popularity and suitability for text

processing.

The rest of the paper is organized as follows: In the next section (Section 2), the problem of direct application of Icon to Chinese text processing is outlined. This is followed by the design of the Chicon language in Section 3. Section 4 presents a series of experiments for evaluating the performance of Chicon. Finally, Section 5 concludes.

2 Icon for Chinese Processing

As mentioned before, the design of Chicon was based on the Icon programming language. Therefore, before it was carried out, the existing problems faced by Icon in Chinese text processing were studied. To achieve this goal, an empirical approach was adopted. A classical Chinese processing application, namely word segmentation [9,10] was developed using Icon (see also Section 4). Word segmentation is the process of identifying word boundaries from a Chinese running text. The lack of clear inter-word delimiters in Chinese text renders word segmentation difficult. This is a problem unfamiliar to Latin based languages, such as English, where words within text are clearly separated by a space. This application was chosen because it involved extensive text manipulation. The study revealed that the following Icon built-in functions were problematic in handling Chinese and Chinese/English mixed texts:

2.1 Built-in Function upto()

The Icon built-in function upto(c,s) finds the character positions in the text string where the character c occurs. For example, `upto('a', text)` where *text* = "I am a boy." returns {2,5}, i.e. the positions just before 'a'. The problem of upto() in handling Chinese is that Icon has no notion of double byte characters. Hence, all the Chinese characters are mistakenly treated as two separate single bytes. This often leads to unexpected results. In the word segmentation application, two sets of tests were performed. upto() was used to locate the words '的' and '地' from texts of different size and different Chinese/English mixture, i.e. using the calls `upto('的',text)` and `upto('地',text)`, respectively. The Chinese characters, '的' and '地', were chosen for the tests because they were two of the most common Chinese characters. The results of the two sets of tests are summarized in Figure 1. More details are also tabulated in Table 1 and Table 2.

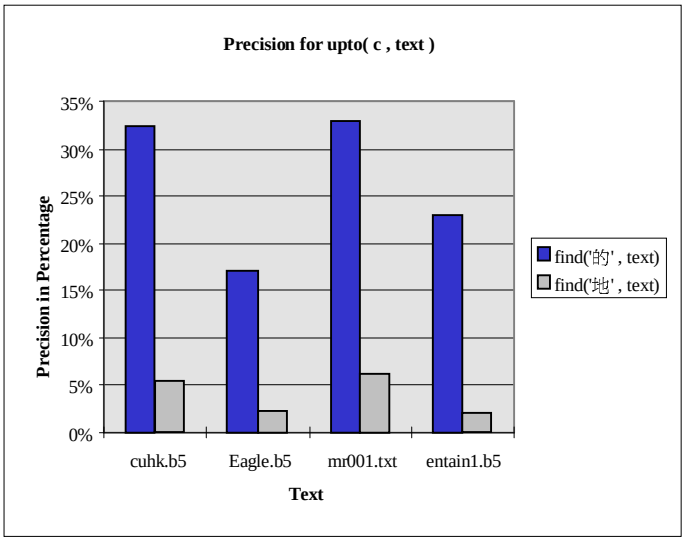

Figure 1: Precision of upto() in Chinese Text Processing.

Table 1: Precision of the Icon Built-in Function — upto('的',text).

Text	No. of Chinese characters (Including punctuation)	Found	Correct hit	Hit Rate (correct/found)
cuhk.b5	1,079	102	33	0.323
Eagle.b5	64,3341	44,760	7,715	0.172
mr001.txt	1,277	156	51	0.33
entain1.b5	582	61	14	0.23

Table 2: Precision of the Icon Built-in Function — upto('地',text).

Text:	No. of Chinese characters (Including punctuation)	Found	Correct hit	Hit Rate (correct/found)
cuhk.b5	1079	110	6	0.0545
Eagle.b5	643341	45495	1044	0.023
mr001.txt	1277	79	5	0.063
entain1.b5	582	47	1	0.0212

From Figure 1, one can observe that the precision rate of using upto() to find a Chinese character from the text is very low. This is because the Icon built-in function upto() finds all the characters in the text which match either the first or the second byte of the Chinese characters of the search string. For example, the BIG-5 representation of '的' in hexadecimal is 0xAABA. The upto() function found all the character positions which contained the hexadecimal code 0xAA or 0xBA. Consequently, `upto('的',text)`would mistakenly return the positions of the upper byte of '服' whose code is 0xAA41 and the lower byte of '征' whose code is 0xA9BA if both characters exist in the *text*.

2.2 Built-in Function find()

The Icon built-in function find(s1,s2) searches for the sub-string s1 in the text string s2 and returns all the matched positions. For example, `find("an", text)` where *text* = "A man with a pan and a fan." returns {4,15,18,25}.

Direct application of find() to Chinese text is erroneous. This is shown in the following example. Consider the invocation of `find("”", text)` in the text containing the string "“滿足”" --- i.e. 0xA1A7 (“) 0xBAA1 (滿) 0xA8AC (足) 0xA1A8 (”) --- at the character position 18, i.e. 0xA1 starts at position 18, 0xA7 at 19, and so forth. In this example, the function returns two integer positions, i.e., {21, 24}. This is because the hexadecimal representation of "”" (Chinese closed double quote) is 0xA1A8 while "滿足" is 0xBAA1 0xA8AC. Thus the function matches the middle two characters of "滿足". There were plenty of examples of this kind of error in the word segmentation test application. The cause of these errors is that the function searches the Chinese text byte by byte while Chinese characters are represented in double-bytes.

2.3 Built-in Function reverse()

As its name suggests, the Icon built-in function reverse(s) reverses the text string s. For example, `reverse("I am a boy")` gives a resultant string of "yob a ma I".

Once again due to the double byte feature of Chinese character, this function cannot achieve its objective on text strings containing Chinese. Consider the string "我是中國人", with the code 0xA7DA 0xAC4F 0xA4A4 0xB0EA 0xA44A. The reversal of the string was expected to be "人國中是我" with the code 0xA44A 0xB0EA 0xAC4F 0xA7DA. However the function simply reversed the string byte-by-byte and resulted in a meaningless byte stream, namely 0x4AA4 0xEAB0 0xA4A4 0x4FAC 0xDAA7.

2.4 Built-in Function sort()

The Icon built-in function sort(X) produces a list containing values from X. If X is a list, record, or set, sort(X) produces the values of X in sorted order. For example, `sort((12,6,3,4))` results in (3,4,6,12).

In the word segmentation test application, sorting is widely used in dictionary management [9]. In practice, sorting in Chinese is essential in nearly all kinds of database applications. Icon can only sort by the internal code order. This is acceptable in English as the ASCII code sequence reflects the alphabetical order. But in Chinese, sorting by internal code is purely for the convenience of computers. It is unnatural. For example, in a paper dictionary for primary school children, words are typically sorted by stroke count. In addition to that, other forms of natural sorting orders in Chinese are: by radical and by PinYin. The lack of support of a different sorting order renders Icon ineffective for Chinese processing.

2.5 Problems in Other Functions

There are many other Icon built-in functions that are incompatible with Chinese characters. Although they were not used in the word segmentation test application, their deficiency in Chinese text processing is obvious. This is mainly due to the fact that Icon was never designed to cope with non-single byte characters. These functions are listed below. In each of them, an example of the error resulted in the application of the function to Chinese is also given.

any (c,s) — locate initial character

If the first character of the string *s* is in the cset *c*, *any()* produces the position after the character. Otherwise, it fails. For example,

```
any('AEIOU',"I am ...")
```

produces 2 since the character at position 1, "I", is in the cset 'AEIOU'. Note that in Icon, a single quote literal represents a cset while a double quote literal represents a string. The following example shows a problem of Icon due to its inability to handle Chinese characters. In Icon, the expression

```
any('一',"二三")
```

gives the result of 2 because both "一" and "二" contain the same character (0xA4) as their most significant byte. To make it clear, a well spaced hexadecimal form of the expression is given below. Note that this is not a valid representation in Icon.

```
any('0xA440',"0xA447 0xA454")
```

The correct result should be a failure as the two Chinese characters "一" and "二" are clearly a mismatch.

many (c,s) — locate many characters

From the input string s, locate the first sub-string which contains one or more of the characters specified in the cset *c*. *many()* returns the character position after this sub-string. It fails otherwise. For example,

```
many('AEIOUaeiou',"Our conjecture has support")
```

produces 3 since the characters in the sub-string "Ou" are members of the cset 'AEIOUaeiou'. Direct application of many() to Chinese does not work. Consider the following:

```
many('一二',"一二三")
```

i.e. `many('0xA440 0xA447',"0xA440 0xA447 0xA454").` The correct result should be 5, i.e. the position just after "一二". However, Icon would produce 6 because it regarded the fifth byte (0xA4), i.e., the first byte of 三, as being in the cset.

match (s1,s2) — match initial string

If the string *s1* occurs in the string *s2*, *match()* returns the ending character position of s1 in s2. For example,

```
match("The","The sky is blue.")
```

results in 4 since "The" occurs in the second string argument. Consider the following Icon expression:

```
match("迨","五六七")
```

i.e. `match("0xADA4","0xA4AD 0xA4BB 0xA443",2).` Icon would try to match "迨" (0xADA4) with the string composing of the least significant byte of "五" (0xA4) at position 2 and the most significant byte of "六" (0xA4) at position 3. Since their binary representations are the same, this would be a hit and the result would be 4. This is incorrect. Instead, a failure would result in this example.

map (s1,s2,s3) — map characters

Given a string s1 that contains the characters described in the string s2, this function replaces the characters specified in s2 with the corresponding characters in the string s3. Note that characters that do not appear in s2 are not changed. For example:

```
map("a+x+y","xy","XY")
```

produces the string "a+X+Y", i.e. "x" and "y" are mapped to "X" and "Y", respectively. Errors occur when map() is applied to Chinese strings. Consider the following example in Icon:

```
map("一二三","一","壹")
```

i.e. `map("0xA440 0xA447 0xA454", "0xA440", "0xB3FC").` The result would be the string "壹蛤袍" (i.e., "0x`B3FC 0xB347 0xB354`"). It resulted because the two bytes of "一" (0xA440) were replaced by the two in "壹" (0xB3FC), and also the first byte of both "二" and "三" (0xA4) was replaced by the first character of "壹" (0xB3). This result is unacceptable. The correct result should be the string "壹二三" .

center (s1,i,s2) — position string at center

It produces a string of size *i* in which *s1* is positioned at the centre, with *s2* used for padding on the left and right sides if necessary. If the size of *s1* is greater than *i*, excessive characters are omitted. *s1* is truncated at the right. The following is an example:

```
center("Icon",9,"+-+")
```

produces the string "+-Icon+-+". Let's look at the effect of center() for Chinese strings. Consider the following example:

```
center("甲乙丙",12,"*")
```

i.e. `center("0xA5D2 0xA441 0xA4FE",12,"0xA1AF")`. The result of this Icon expression is "* 猖A丙砥 " (i.e., "0x`A1AF 0xA1A5 0xD2A4 0x41 0xA4FE 0xAFA1 0xAF`"). In producing the resultant string, for half of the Chinese character in s1, "*" is used for padding. As a result, the whole content is corrupted. The correct result should be "* 甲乙丙 *". Note that other string formatting built-in functions, such as right() and left() suffer from a similar error.

2.6 The Problems

Summarizing from the preceding discussion, one can identify a number of problems in the direct application of Icon to Chinese text processing, viz:

- the string matching problem, commonly referred to as the misalignment problem [7] (see upto() and find()); and
- the character position problem (see upto(), find() and reverse()); and
- the sorting problem (see sort())

The above problems are originated from the semantic of the Icon language.

3 Design of Chicon

Since Chicon is based on Icon, it is essential in the design of Chicon not only to overcome the problems mentioned in Section 2, but also to be compatible with Icon. As such an application program written in Icon can be readily compiled by the Chicon compiler. The Chinese processing capabilities of Chicon were designed in three stages:

Stage 1: Definition of Chinese data types.

Stage 2: Type conversion mechanism between Chinese and existing text data types.

Stage 3: Based on the new Chinese data types, design a new set of built-in Chinese text processing functions.

3.1 Chinese Data Types

3.1.1 Chicon supported Characters

Chicon was designed to cope with texts comprised of both English and Chinese characters. Chinese characters are represented either in double byte BIG-5 or GB and English in single byte ASCII. In addition to the difference in code size, the most significant bit of a Chinese character is always set to '1'. On the other hand, ASCII is

assumed to be a 7-bit entity with the most significant bit always zero.

In addition to ASCII characters (7-bit), Icon can handle single byte non-ASCII characters (8-bit). In order to be compatible with Icon, Chicon must also cope with this situation. However, this would complicate the process of differentiating Chinese and English characters. The small program shown in the introduction would no longer work. For this reason, a new string type was introduced in Chicon (see below).

3.1.2 Binary String and Text String

A string is comprised of a list of characters. In Icon, any number of single-byte, including both ASCII (7-bit) and non-ASCII (8-bit), characters can appear within a string. This is kept in Chicon and the data type is referred to as *binary string*. To account for English/Chinese mixed text, a new data type, namely *text string*, was introduced. The latter is comprised of a list of 7-bit English (ASCII) and 16-bit Chinese characters. In other words, a text string only includes printable characters and a binary string contains both printable and non-printable ones. Another major difference is that each element in a binary string is treated as single-byte entity while the size of an element in a text string varies depending on whether it is a Chinese or an English character.

Text strings enable Chicon to bypass the problem of differentiation between 8-bit non-ASCII and Chinese characters. Notice that a string consisting of non-ASCII and Chinese mixed is not supported in Chicon. This is acceptable in practice for such strings rarely occur; and in circumstances where a non-ASCII character is required to insert in a Chinese text string, its 16-bit BIG-5 or GB equivalence can be used.

It is noteworthy that a binary string that contains only printable (7-bit ASCII) characters degenerates to a text string consisting of English characters. Thus, all existing Icon programs that process pure English text can be executed in the Chicon execution engine without any modifications. This renders Chicon compatible with Icon.

String subscript. Traditionally, when dealing with English plain text consisting of only 7-bit ASCII characters or binary data containing 8-bit non-ASCII characters, a character in a string is referenced by a number indicating its physical distance, or the number of bytes, from the beginning of the string. Application of this convention to Chicon would work only with binary strings; but it would be unsuitable for text string because of the existence of Chinese characters. Consider the example "CU 的全名是 Chinese University" (i.e. the full name of CU is Chinese University). In Icon, the position of C in "CU" is 1 and the position of C in "Chinese" is 13 (NB: space is counted as one character). Clearly, this way of string subscripting is unnatural and misleading. To overcome this, a more natural string subscripting policy was introduced to Chicon. Under this policy, position in a string is counted by characters. Applying it to the above example, the positions of the C's in the sentence are 1 and 9, respectively.

Furthermore, based on this definition, Chicon can more meaningfully work out the size of a string. Take the above as an example, in Icon, the length of the strength is counted by the number of bytes and is equal to 30; on the other hand, in Chicon, the

length is counted by the number of text characters and is equal to 16.

3.1.3 Cset Extension

Cset (i.e. character set) is a special data type in Icon. It is an unordered set of characters useful for various kinds of text manipulation. Semantically, a cset is comprised of disjunctive collection of characters. For example, the cset 'abc' represents any one of the characters 'a', 'b' or 'c'. It is widely used in many Icon built-in functions, e.g. any('aeiou',"i am a boy") can detect whether the first character in the string "i am a boy" is a vowel. Unfortunately, the original *cset* in Icon only supports 8-bit single-byte characters and there is no way to include Chinese characters into it. For this reason, the definition of cset was extended in the design of Chicon.

In designing the cset data type for Chicon, two different definitions have been considered. The first one required that 2-byte coded characters and non-ASCII single-byte characters to be mutually exclusive. As such the use of a cset would be restricted to either text or binary data. This would enforce strong typing and make csets awkward to use. For example, the union of a cset containing 2-byte coded characters and another cset containing non-ASCII single-byte characters would lead to an undefined cset.

The other definition of cset, which is adopted by Chicon, allows the co-existence of text characters and single-byte characters. Under this definition, results of most of the operations involving csets are well defined, and no type conversion is required. The usage of the extended cset in Chicon is the same as using csets in Icon. Any entry in a cset, either a single-byte or a 2-byte character, can match the same character in a string. Consider the example,

```
any('我你他',"他是學生")
```

The above expression checks whether the first character of the string appearing in the second argument is in the cset specified in the first argument, i.e., it checks whether the first character of the string "他是學生" is any of '我', '你' and '他'.

Since strings in Chicon are clearly classified as either text or binary strings, a 2-byte coded character in a cset never matches with any character in a binary string. Similarly, a single-byte character in a cset never matches with half of a 2-byte coded character in a text string. For example, consider the Chinese character "你" (0xA741) whose lower byte is equivalent to the ASCII character 'A' (0x41). If one wants to find the position(s) of the character 'A' in a sentence by invoking:

```
upto('A', "你是JASON嗎？")
```

Using Icon, positions 3 and 7, which are the positions after the hits, would be returned from the above expression. On the other hand, using the extended cset in Chicon, only position 5 would be resulted.

3.1.4 Quoted Literals

Quoted literals include cset literals and string literals. In Chicon, cset literals are represented as a collection of characters embraced in single quotes and string literals are the same in double quotes. Since these literals may contain both text and single-byte characters, some rules were defined in the Chicon compiler to distinguish between them.

Customarily, people seldom type non-ASCII single-byte characters directly into a program since they are normally invisible. The design of Chicon for interpreting quoted literals was based on this heuristic. Any non-ASCII character in a quoted literal is regarded as part of a 2-byte coded character. If the Chicon compiler encounters a non-ASCII character without any following characters or if it cannot form a 2-byte coded character together with the character following it, a syntax error is reported. For example, the Chicon expression `write("0xA1"),` where "0xA1" is an 8-bit character, gives rise to a syntax error.

To include non-ASCII single-byte characters in quoted literals, escape sequences must be used. An escape sequence consists of a backslash followed by one or more characters. Two of them are of interest. They are *ddd* and *xdd*. The sequence *ddd* stands for the single-byte character with octal code ddd, where d is an octal digit. The sequence *xdd* stands for the single-byte character with hexadecimal code 0xdd, where d is a hexadecimal digit. Uppercase and lower case hexadecimal digits are equivalent. A character specified using an escape sequence is always a single-byte entity.

For string literals, the string type (see Section 3.1.2) must be determined. The Chicon compiler makes this decision based on the rule that once an escape sequence with the form of \|*ddd* or \|*xdd* occurs in a string literal, the string is considered as binary; otherwise a text string is assumed. For example, the string literal "CU 工程" in a Chicon program is stored as a text string whereas the literal "CU\x20工程" (note that 0x20 is the ASCII code of the space character) is stored as a binary string. The same concern is inapplicable to cset literals as both single and double bytes cannot coexist in a cset (see Section 3.1.3).

3.2 Type Conversion

Similar to Icon, type conversion is possible among strings, csets and numerics in Chicon. There are two forms of conversion: explicit and implicit. Implicit type conversion takes place when the data type of an argument to a function or an operand given to an operation is different from what the function/operation expects. Impossible conversion results in run-time error. Explicit conversion is performed using the built-in functions string(), bstring(), cset(), bcset(), integer(), real() and numeric(). The last three have the same semantics as the same defined in Icon. However, string() and cset() in Icon were found inadequate to support all types of conversions in both textual and binary data. For this reason, the built-in functions, bstring() and bcset() were introduced to Chicon. The prefix "b" stands for binary operations. Conversions to strings can be accomplished with string() and bstring(); and similarly, conversions to csets can be done by using cset() and bcset().

3.2.1.1 string() and bstring()

string() converts a cset to a text string. If the argument is a cset containing ASCII and 2-byte coded characters, the resulting string will have ASCII characters at the front followed by 2-byte coded characters ordered by their ordinals. For example,

```
string(',.，。')
```

where ',' is a comma; '.' a full stop, '，' a 2-byte encoded comma and '。' a 2-byte encoded full stop. The above expression results in the text string ",.，。". If the argument is a cset with non-ASCII single-byte characters, the conversion is impossible since a text string does not allow any non-ASCII single-byte character and hence the function call fails. For such a case, bstring() should be used instead.

bstring() accepts any kind of cset and converts the argument to a binary string. The resulting binary string is comprised of a sorted collection of the single-byte characters appear in the cset and the single-byte characters that comprise the 2-byte coded characters in the cset.

3.2.1.2 cset() and bcset()

The function cset() takes a string as its sole argument and converts it to a cset. Digits, symbols and single-byte characters produce single-byte character entries and Chinese characters produces 2-byte coded character entries.

bcset() is used to convert the argument which is a string to a cset with only single-byte characters. During this conversion, any 2-byte coded character is treated as two single-bytes. Therefore, no 2-byte coded character exists in the resulting cset. For instance, the expression:

```
cset(",.，。")
```

results in a cset containing four elements, i.e. ',', '.', '，' and '。'. On the other hand,

```
bcset(",.，。")
```

results in a cset containing five members, i.e. '!', '#', ',', '.', and a non-ASCII single-byte character 0xA1. Note that "!", "#" and 0xA1 are produced from the two 2-byte encoded punctuation symbols, i.e. "，" and "。".

3.3 Built-in Operations and Functions

3.3.1 Lexical Comparison

Unlike English and other Latin languages, there is no alphabet in Chinese. Under a non-Chinese based computing environment, comparison between two Chinese characters is typically based on their internal code. This works perfectly for comparison based on equality. But for comparison involving less than, greater than, ... etc. operations, using internal code may not be sensible. In practice, the comparison is more meaningful if it is based on the number of strokes, the radicals and the Pin-Yin, which is certain kind of phonetics, of the two characters. Furthermore, since there is

no standard in the sorting order of the Chinese character set, other users may prefer to use other criteria for the comparison.

Based on the aforesaid observation, Chicon was designed to provide a flexible means to support multiple criteria for Chinese character comparison. First of all, it was decided not to use built-in symbols, such as "Big5" without the quotes, to specify comparison criteria. This would prevent the programmers from using new comparison criteria defined for their own purpose. This is essential as they may choose to use some non-standard character ordering scheme. For this purpose, a new built-in function, *register(i, order_table)*, was introduced in Chicon. The function associates an integer identifier, *i*, with a comparison criterion defined in *order_table*. For example, `register(4,PinYin_Table)` assigns the *PinYin* comparison criterion with the number 4. Next, a new in-fix binary operator, namely, the comparison criterion operator, was defined. Its usage is as follows:

Identifier $ Expression

where *Identifier* is an integer identifier defined by the users with the built-in function register(); *$* is the comparison criterion operator; and *Expression* is the comparison expression. For example, with the following declaration and registration procedure:

```
PinYin := 4
register(PinYin,PinYin_Table)
```

the expression "`PinYin $ (string1 > string2)`" implies "Is string1 greater than string 2 according to certain phonetic comparison criterion?". In practice, the *Identifier $* part is optional. Without it, the *Expression* assumes the conventional internal code comparison criterion which is either BIG-5 or GB depending on the computing environment.

3.3.2 String Related Built-in Function

The problem of various Icon built-in functions, e.g. upto(), find(), any() and sort(), in handling Chinese text is outlined in Section 2. These functions are no longer problematic in Chicon with the new *text string* data type and the extension in cset (see Sections 3.1.2 and 3.1.3, respectively).

The invocation of sort() in Chicon is worth mentioning. It adopts the same mechanism as lexical comparison (see Section 3.3.1). By default the internal code ordering table (this is synonymous to the comparison criterion table, Section 3.3.1) is used. If the programmer wishes to sort a string by another order, he/she uses the following:

```
i $ sort(string)
```

where *i* is an integer associated to the ith ordering table, $ is the comparison criterion operator (see above) and *string* is the string to be sorted.

3.4 Implementation Issues

The introduction of the new data types as well as various modifications in the execution environment would have slowed down the performance of Chicon. To alleviate this predicament, the following implementation issues have been addressed in the design:

3.4.1 Implementation of Cset

Cset is a data type widely used in a number of Chicon built-in functions, e.g. `any('aeiou',sentence)`. Due to the frequent usage of these functions, the implementation of cset is performance critical.

Implementation of the cset data structure, i.e., a cset block, was divided into two parts. The first part is for the 256 different single byte characters. This part is similar to the cset data structure in Icon [3]. It uses eight 32-bit words to record the presence or absence of each of the 256 characters. Each bit represents one character. This is quite efficient in terms of both time and space.

The second part was designed to accommodate the cset extension for the Chinese characters. It would have been impractical to use the same technique as the first part. Since there are 32,768 possible 2-byte characters with first bit set, 4,096 bytes are required to store all the bits. Such a representation would be too long. From the application studies, it was observed that a cset is usually comprised of fewer than 10 Chinese characters. Thus, it is more space efficient to simply store the characters directly. This approach only requires 20 bytes which is far less than the bit representation approach in Icon. Further, to facilitate fast searching, the Chinese characters in a cset block are sorted by internal code and duplicates are omitted. A membership test can be efficiently achieved using a binary search through the sorted character list.

3.4.2 String Matching

String matching is by far the most widely used function in any text processing programs. In particular, in Icon and Chicon application programs, *generators* [2,3,4,5] are widely used for this purpose. The concept of a generator is best explained by an example. Consider the following set of expressions:

```
sentence := "Store it in the neighboring harbor."
if (i := find("or",sentence)) > 5 then write(i)
```

The second expression shows a typical use of a generator function, i.e. find(). Here, the variable i is first set to 3. Since it is not greater than 5, the same expression is executed again and i is set to 23 which is greater than 5 resulting in the execution of write(i). This example shows that generators can heavily involve string matching. Therefore, it is essential in the design of Chicon to have an efficient string matching mechanism.

After considering various string matching methods, the modified quick search algorithm [7] was adopted. In the initialization phase of quick search, the pattern string is scanned through and a "delta" table is first created. The shift function, as described in [11], makes use of the delta table to determine how far forward to shift

the pattern string when a mismatch occurs. The misalignment problem of applying the algorithm on mixed Chinese/ASCII texts is discussed in [7]. To get around this problem, the shift function was modified. After the number of skipping characters is determined, the characters between the current position and the next position are read from left to right. This determines whether the character at the next position of the searching string is the least significant byte (LSB) of a Chinese character. If it is, moving the pattern to that position will cause a misalignment since the beginning of any pattern is supposed to be an ASCII character or the most significant byte (MSB) of a Chinese character. In the case of a misalignment, the pattern string is moved to the position immediately following the designated one, i.e. the one determined by the original shift function; and then the normal steps of quick search are resumed.

The above string matching mechanism only applies if either the pattern or the text under examination is a text string. Otherwise the original quick search algorithm, i.e. without considering the Chinese character misalignment problem, is used.

3.4.3 String Type Determination

Chicon supports dynamic data typing. The type of a string is automatically determined by the Chicon system when it first appears. Its type can, however, be overridden explicitly in the program using a set of type conversion functions (see above). A Chicon string can first appear in a program as a quoted literal or at run-time as a result of a read operation. The former is handled by the Chicon compiler and the latter the Chicon execution engine. Irrespective of the executor, the following algorithm forms the basis for determining the type of a string:

```
1.  regard input string as a text string by default;
2.  WHILE (not the end of the string)
3.  {
4.     IF (the current byte is a non-ASCII character)
5.     {
6.       get the next byte in the string;
7.       IF (both bytes together do not form a valid Chinese character)
8.       {
9.         regard this string as a binary string;
10.        break the while loop;
11.      }
12.      unget the second byte;
13.    }
14.    current byte := next byte;
15. }
```

In the above algorithm, the Chicon system, either the compiler or the execution engine, first assumes the string is a text string by default (line 1). It then inspects each element in the string byte by byte (lines 2 to 15). If an element is a non-ASCII character (line 4), it will inspect the subsequent element to determine whether it is a valid Chinese character (line 7). If it is not, this string must be a binary string (line 9).

3.4.4 Order Tables for Lexical Comparison

Chicon enables programmers to define their own lexical comparison criteria. This is achieved by using order tables. An order table is an array of numbers, represented as 2-byte words, denoting the ranking of all Chinese characters. Each element in the array is associated with a particular Chinese character and can be located directly by the code of the Chinese character. Since the most significant bit is always set to 1 in BIG-5 and GB, there are total 32768 entries in the array. When two Chinese characters are compared, the elements in the specified order table corresponding to these Chinese characters are located and the numbers stored are compared.

4 Testing and Evaluation

4.1 Application Tests

To assess the robustness and practicality of the language, two string manipulation applications were developed in Chicon. The tests aimed to evaluate the ease of use of Chicon and to compare the program complexity of Chicon with both Icon and "C" for Chinese computing. The two applications were: a Chinese word game and Chinese word segmentation.

4.1.1 A Chinese word game

In this game, a Chinese sentence was randomly divided into a sequence of words and the users were asked to re-construct the sentences from the random word sequence. For example,

校車　各處。　校園　有　穿梭　中大

is the word sequence generated from the following Chinese sentence:

中大　有　校車　穿梭　校園　各處。
(The Chinese University has shuttles running around the campus.)

The structure of the simple game is divided into five parts:

1. Prompt the user and ask for the user's preference of difficulty;
2. Read the database that contains a database of sentences;
3. Randomly select a sentence and generate a sequence of words from it;
4. Display the word sequence;
5. Re-arrange the words in the sequence (by the users); and
6. Record the user's answer and show the correct (i.e. the original) sentence.

The number of statements in the word game programs written in Chicon, Icon, and "C" are 127, 128 and 271, respectively. Further, looking into the three programs in detail, it was observed that the Chicon code was in general the simplest, especially in the parts that involved string manipulation. To further reveal the relative complexity of the three languages, a larger application, namely word segmentation, was used.

4.1.2 Word segmentation

Word segmentation is a process to identify "words" from a Chinese sentence [10]. It forms the backbone of almost every Chinese natural language processing systems. This application is large in size and complex in nature. It can, therefore, provide a reliable indication of the robustness and practicality of Chicon.

The Chicon word segmentation program consisted of 853 lines with (a) 28 functions modules, (b) a large number of control structures — 35 *while* loops and 58 *if* expressions, and (c) various high level string processing expressions such as generators. The same application was developed in Icon. The size of the target code generated by Chicon was smaller than Icon despite the fact that Chicon is more powerful and is capable of handling both English and Chinese text.

In addition to its efficiency in Chinese processing, programming in Chicon was relatively simple compared to Icon and C. The following 2 examples highlight this point:

Find the Chinese character set: In order to search for a pattern comprised of a Chinese character set using Icon, constant checking is required to ensure the correct results. In the Icon program below, the string s2 is searched to look for the cset s1. Undoubtedly, it is long and clumsy.

```
1  procedure ch_upto( s1, s2, i1, i2 )
2      if s2 === &null then {
3            s2 := &subject
4            i1 := 1
5      } else {
6            if i1 === &null then  i1 := 1
7      }
8      if i2 === &null then  i2 := 0
9      i := 1
10     while i <= *s2 do {
11           j := 1
12           while j <= *s1 do {
13                 if s2[i]==s1[j] & (ord(s1[j])<128 |
14                       s2[i+1]== [j+1] ) then {
15                       suspend i
16                 break
17                 }
18                 if ord( s1[j] ) > 127 then  j +:= 2
19                 else j +:= 1
20           }
21           if ord( s2[i] ) > 127 then i +:= 2
22           else i +:= 1
23     }
24  end
```

On the other hand, the same in Chicon simply involves a function call, viz:

```
upto(s1,s2,i1,i2)
```

Reverse a sentence: It is very simple using Chicon to handle Chinese text or a mixture

of Chinese text and binary strings. For example, if one wants to reverse a sentence comprised of both Chinese characters and ASCII characters in Icon, both Chinese characters and binary strings have to be explicitly detected. For this reason, the program is very complicated as shown in the following example.

```
1  procedure rev_sent_div()
2      j := 1
3      k := 1
4      sent_start := 1
5      sent_end := 3
6      temp := list(200)
7      while j < sent_count do {
8      rev_sent[j] := ""
9      while sent_end <= *sent[j] + 1 do {
10                #check if it is the Chinese character #
11                if ord(sent[j][sent_start]) >= 161 then {
12                      temp[k] :=
13                            sent[j][sent_start:sent_end]
14                      k +:= 1
15                      sent_start +:= 2
16                      sent_end +:= 2
17                } else {
18                      temp[k] :=
19                            sent[j][sent_start:sent_end-1]
20                      k +:= 1
21                      sent_start +:= 1
22                      sent_end +:= 1
23                }
24           }
25           k -:= 1
26           while k > 0 do {
27                rev_sent[j] ||:= temp[k]
28                k -:= 1
29           }
30           j +:= 1
31           sent_start := 1
32           sent_end := 3
33           k := 1
34     }
35  end
```

Using Chicon, the program is simpler since both text and binary strings can be automatically detected. For example, the program shown below is used to reverse each of the elements in the list *sent*, and assigns the results to another list *ev_sent*.

```
1  procedure rev_sent_div()
2      j := 1
3      ev_sent := list( *sent )
4      while j < *sent do {
5            ev_sent[j] := reverse(sent[j])
6            j +:= 1
7      }
8  end
```

The above two program fragments clearly show that the code size of the Chicon version is less than that in Icon. Furthermore, since the string matching modules in the word segmentation process frequently requires finding a cset and reversing a string, it was not surprising to observe that the Chicon program ran faster than its Icon counterpart.

4.2 Performance Evaluation

A series of experiments were conducted to study and compare the performance of Chicon and Icon. The experiments were carried out on SUN SPARCstation 5 under the Solaris 2.3 OS.

A pure ASCII file was used to evaluate the efficiency of a number of string-related built-in functions — *find()*, *reverse()*, *sort()*, and *sortf()*. The testing file was 201K bytes in size and consisted of 5,010 lines. Detailed descriptions of, and results collected from each experiment are as follows:

find(): Two different cases were designed for assessing the efficiency of the built-in function *find()*. The first case was prepared to evaluate the efficiency of the function *find()* on pure ASCII strings or binary strings. The second case was designed to evaluate the efficiency on pure text strings.

Case 1: "*find("the" , S1)*" — find the string *"the"* in *S1*, where *S1* was a string of 203K bytes in size — was used to assess the efficiency of the built-in function *find()* on pure ASCII strings. The result for 100 iterations of this test is summarized in Table 3.

Case 2: "*find("的" , S2)*" — find the string *"的"* in *S2*, where *S2* was a string of 1,468K bytes in size. The result of the test is shown in Table 3.

Table 3: Average Execution Time for *find()*.

Average Time (sec.)			
Cases	Icon	Chicon (text string)	Chicon (binary string)
1. find("The", S1)	33.51	5.89	2.92
2. find("的", S2)	72.69	0.28	0.19

Case 2 only measured the time to find the first appearance of 的 in the text string S2. To extend this, a test to find all appearances of 的 in the text string S2 was carried out. In Chicon, this test was specified as a single expression, i.e. "*every find("的",S2)*". On the other hand, a special Icon routine was written in order to achieve the same goal (see Figure 2). The average execution time using Chicon was 55.24 seconds. It was 24% faster than Icon. The main reason that Chicon outperforms Icon in find() is because the former employs the modified quick search algorithm [7] (see also Section 3.4.2).

```
1.  procedure main()
2.          …
```

```
3.          every ch_find( "的" , S ) # find all "的" in S
4.          …
5.  end
                …
1-1. procedure ch_find( s1, s2, i1, i2 )
1-2.          if s2 === &null then {
1-3.               s2 := &subject
1-4.               i1 := 1
1-5.          } else if i1 === &null then  i1 := 1
1-6.          if i2 === &null then  i2 := 0
1-7.          l := *s2
1-8.          i := 1
1-10.         while i <= l do {
1-11.         if match( s1, s2, i, i2 ) then suspend i
1-12.              if ord( s2[i] ) > 127 then  i +:= 2
1-13.              else i +:= 1
1-14.         }
1-15. end
```

Figure 2: Icon routine to achieve every *find("的" , S).*

reverse(): *every reverse(!L)*, which reverses every element of the list !*L*, was used to test the efficiency of the reverse() function. The list !*L* used was 203K bytes in size and consisted of 5,010 elements of strings. The average execution time of 100 iterations for the test is summarized in Figure 3.

sort(): *sort(L)*, which returns the sorted list of *L*, was used to test the efficiency of the sort() function. Like *reverse()*, the list *L* used was 203K bytes in size and consisted of 5,010 elements of strings. The average execution time of 100 iterations for the test is also summarized in Figure 3.

sortf(): *sortf(L,i)*, which returns the sorted list of *L*, was used to test the efficiency of the function. Different from sort(), the input to sortf() is a multi-dimension list, e.g. {{1,20,3},{7,9,10}} and an integer identifier. The latter determines which element of the embedded list to sort on, e.g. `sort({{1,20,3},{7,9,10}},2)` results in the list {{7,9,10},{1,20,3}}. Like *reverse()* and *sort()*, the list *L* used for the experiment was 203K bytes in size and consisted of 5,010 elements of strings. The average execution time of 100 iterations for the test is also summarized in Figure 3.

upto(): *upto('aeiouAEIOU' , S)*, which returns all the positions of the character set 'aeiouAEIOU' in the string *S*. The string *S* used was 203K bytes in size and was composed of pure ASCII characters. The average execution time of 100 iterations for the test is also summarized in Figure 3.

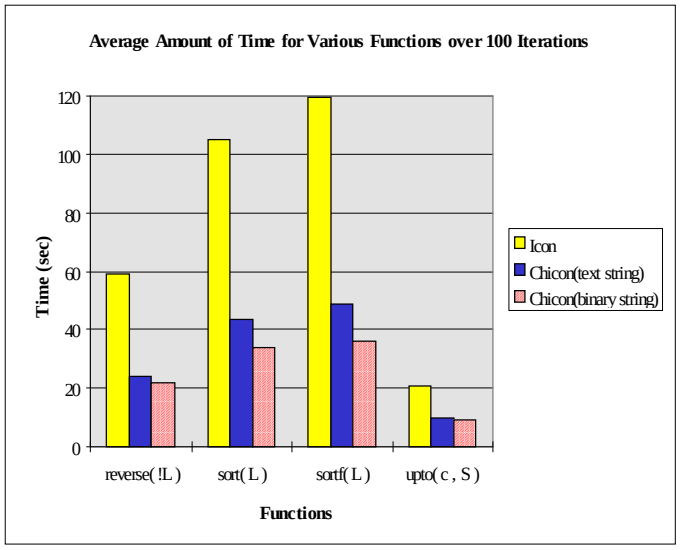

Figure 3: Comparison of Performance on String-related Functions.

From Figure 3, it is obvious that the target codes generated by Chicon are quite efficient. It is, therefore, evident that Chicon is superior to Icon in string related operations for Chinese.

5 Conclusions

Existing text manipulation languages, e.g. Icon, are unsuitable for development of Chinese information systems. It suffers from low efficiency as well as high error rate. For this reason, Chicon, a language based on the text manipulation concepts of Icon and with built-in Chinese processing capability, was introduced. At the programming level, Chicon provides new data types, i.e. text string and cset extension, for representing Chinese text. At the implementation level, an enhanced quick search algorithm, which supports fast Chinese/English mixed text searching [7], was adopted. A series of evaluation tests have shown that Chicon out-performed Icon in Chinese text manipulation. The advantage of Chicon over Icon is not limited to performance. Its code, in general, is more compact for Chinese computing applications. Furthermore, the development of the word segmentation and word game programs using Chicon has proved that Chicon is a practical programming language.

To extend the practicality of Chicon, the possibility of coupling Chicon to different languages like Java and SQL is being studied. Although Java is a popular programming language, its text manipulation facility is not so strong, especially for Chinese texts. It is envisaged that the coupling of Chicon with Java will open up many new web-based applications.

Acknowledgement

This project is partially funded by the Hong Kong Research Grant Council under the 1994/95 Earmarked Grant initiative (no. CUHK 256/94E). Thanks are due to Michael Gordon, T.J. Parr, Margret Newman, Greg Townsend, Ken Walker and Cliff

Hathaway of the Icon research team, Department of Computer Science, University of Arizona, Tucson, for providing us many valuable suggestions and help in the design of Chicon. We are also indebted to Hung Chiu_Hung, Jason Cheung and Louis Ngai who have participated in the development of the Chicon programming system and the sample applications.

References

[1] Griswold, R.E., *The SNOBOL 4 Programming Language*, 2nd Ed. Englewood Cliffs, N.J., Prentice Hall. 1971.

[2] Griswold, R. E. and M. T. Griswold. *The Icon Programming Language*, Englewood Cliffs. N.J., Prentice-Hall, Inc. 1983.

[3] Griswold, R. E. and M. T. Griswold, *The Implementation of the Icon Programming Language*, Princeton, N.J., Princeton University Press. 1986.

[4] Griswold, R. E. and M. T. Griswold. *The Icon Programming Language*, Englewood Cliffs. N.J., Prentice-Hall, Inc. 2nd Ed. 1990.

[5] Griswold, R. E. "An Overview of the Icon Programming Language, Version 8*", TR 90-6d. Computer Science Department, the University of Arizona. 1991.

[6] ----, *Code of Chinese Graphic Character for Information Interchange*, Primary Set (GB2312-80), National Standards Bureau, Beijing, China, 1990.

[7] Wong, K. F. "String Matching on Chinese/English Mixed Texts", *International Journal of Computer Processing of Chinese and Oriental Languages*, Vol10, No.1, published by Chinese Language Computing Society, June 1996, pp115-126.

[8] ISO, “Information Technology - Universal Multiple-Octet Coded Character Set (UCS)”, Part 1: Architecture and Basic Multilingual Plane, ISO/IEC10646-1: 1993, ISO, Geneva, 1993.

[9] Wong, K.F., Lum, V.Y., and Leung, C.H., "Parallel Chinese Word Boundaries Identification in the IPOC Information Retrieval System", *International Journal of Information Technology*, vol 3(1), World Scientific Publishing Co., June 1997, pp63-81.

[10] Lui, Y. et. al., *Word Segmentation Standards*, Tsinghua University Press, Beijing, China, 1994. (in Chinese)

[11] Hume, A., and D. Sunday. "Fast String Searching". *Software Practice and Experience, 21 (11)*. Nov 1991. pp. 1221 - 1248