Integrating pattern data types within Unicon string scanning

John H. Goettsche
Dept. of Computer Science
University of Idaho

July 29, 2014

Abstract

1 Introduction

In order to enhance their productivity in scanning and analyzing strings, string scanning and pattern matching systems have been developed. Modern high-level languages tend to offer reqular expressions and context free grammars for their string processing. Analysis of strings often require functionality beyond what these classes of languages offer. SNOBOL4 was the most successful of the early string processing languages. [1] The pattern data type was imployed in SNOBOL4 along with pattern operators and functions to perform the analysis.

A successor to SNOBOL4 was Icon, whicn expanded upon the goal directed evaluation with generators and backtracking that was implicit in SNOBOL4's pattern matching. [1] It uses a string scanning method which passes through the subject string with a set of commands to manipulate and analyze its contents. Many icon programmers have expressed a desire for the functionality of SNOBOL4 patterns. [2].

When Unicon was developed, its core elements came directly from Icon, where it inherited its string scanning functions. [5] Many of the Unicon string scanning functions may resemble SNOBOL4 patterns, but they are actually different in both how they processed as well as their functionality. Patterns are defined data types that are used deductively to test whether the pattern exists within a subject string, while a string scanning uses a set of functions to inductively analyze or extract data from the subject string. The difference being that patterns are pre-defined and applied later, while string scanning is performed as they are executed.

This paper briefly explores the background of SNOBOL4 patterns and Unicon string scanning functions, a proposed design considerations of implementing SNOBOL4 patterns within the Unicon string scanning environment, and description of how that implementation can be accomplished.

2 Background

SNOBOL4 was developed by Bell Telephone Laboratories in 1962. Searching for a desired pattern within a string of characters is one of its basic operations. Patterns could be as simple as a single character or a set of characters in a particular order, or it can be a complex arrangement with alternative character sets and pattern function. The pattern data type was used enabling the user to define and store patterns as variables to be used later when they were desired. [4]

One of the developers of SNOBOL4, Ralph Griswold, went to the University of Arizona and developed the Icon programming language which was more readable and simpler to use. [5] Unlike most other languages Icon considers the string as a data type in its own right, rather that a set of characters. [3] The string scanning environment allows the user to execute a variety of string functions to search for sub-strings or patterns.

Using the same Icon source code, Unicon was developed to include modern software features such as objects, networks and databases. [5] The string scanning environment of Icon is a part of the Unicon programming language. Master's student, Sudarshan Gaikaiwari proposed adopting SNOBOL4 patterns to the Unicon language. In his thesis, he added the pattern data type, and provided pattern matching functions to execute the pattern searches. [1] The pattern data type was kept separate from the string scanning environment. This paper proposes the pattern matching proposed by Gaikaiwari be refined to be more naturally incorporated in the Unicon language with the string scanning environment implement the pattern matching functionality.

3 Design Considerations

3.1 Pattern matching statements

The pattern matching statement in SNOBOL4 and Gaikaiwari's Unicon implementation are in the following statements:

SNOBOL4:

```
SUBJECT  PATTERN
```

Unicon:

```
subject ?? pattern
```

In the above lines, the subject would be scanned to see if it contains the contents of the pattern, if it succeeds, then a substring of the subject that fits the pattern would be returned. With the SNOBOL4 operation the pattern has to begin its match at the first character of the subject string in the anchored mode; in the non-anchored mode the pattern could start at any character in the string. [4] The Unicon operation is in the non-anchored mode by default. [1]

The Unicon string scanning evironment is initialized in the following statement:

```
subject ? expr
```

The '?' operator sets the cursor location to the first position in the subject string and the function or the block of functions that are called in the expression are executed. Each function moves the cursor upon success, for this reason, I propose if in the event that a pattern is encountered with the tabmat '=' operator, the pattern match be performed in the anchored mode, with the cursor being advanced to the end of the matching pattern if there is success.

3.2 SNOBOL4 and Unicon pattern operators

Table 1: Pattern Operators

Operation	SNOBOL4	Unicon
Concatenation	implicit	||
Alternation	|	.|
Immediate Assignment	$	$$
Conditional Assignment	.	−>
Cursor Assignment	@	.$
Unevaluated Expression	∗x	‘x‘

3.3 SNOBOL4 and Unicon pattern functions

The table below shows the SNOBOL4 primitive functions and the recommended Unicon pattern functions. In most cases it is recommended that the function be lexically similar with the first character being capitalized and the following letters in lower-case, with a couple of exceptions for FAIL and ABORT. Fail appears to be already used in Unicon, therefore PFail is being recommended to represent Pattern Fail. Abort is being changed to Cancel to represent the cancellation of the entire matching operation. This results in the following set of Unicon pattern functions in comparison to their original SNOBOL4 pattern functions:

Table 2: Pattern Functions

SNOBOL4	Gaikaiwari	Unicon
LEN(n)	PLen(n)	Len(n)
SPAN(c)	PSpan(c)	Span(c)
BREAK(c)	PBreak(c)	Break(c)
ANY(c)	PAny(c)	Any(c)
NOTANY(c)	PNotAny(c)	NotAny(c)
TAB(n)	PTab(n)	Tab(n)
RTAB(n)	PRtab(n)	Rtab(n)
REM	PRest()	Rem()
POS(n)	PPos(n)	Pos(n)
RPOS(n)	PRpos(n)	Rpos(n)
FAIL	PFail()	PFail()
FENCE	PFence()	Fence()
ABORT	PAbort()	Cancel()
ARB	PArb()	Arb()
ARBNO(p)	PArbno(p)	Arbno(p)
BAL	PBal()	Bal()

3.3.1 Len

Len(n) function matches a string of characters of n length beginning from the current cursor position. Unicon string scanning uses move(n) for a similar result where the substring of characters of n length are matched from the current cursor position. In non-anchored pattern matching mode, the location of the cursor is not fixed, therefore Len(n) is more appropriate as the substring location has yet to be determined. Some examples are as follows:

SNOBOL4:

```
subString = '1941 Dec. 07'
infamy = LEN(4) . YR ' ' LEN(4) . MO ' ' LEN(2) . DAY
subjectString  infamy
```

Unicon Patterns (non-anchored mode):

```
subjectString := "1941 Dec. 07"
infamy := Len(4) $$ YR && " " && Len(4) $$ MO && " " && Len(2) $$ DAY
subjectString ?? infamy
```

Unicon Patterns (anchored mode):

```
subjectString := "1941 Dec. 07"
infamy := Len(4) $$ YR && " " && Len(4) $$ MO && " " && Len(2) $$ DAY
subjectString ? {
   =infamy
}
```

Unicon string scanning:

```
subjectString := "1941 Dec. 07"
subjectString ? {
   YR := move(4)
   move(1)
   MO := move(4)
   move(1)
   DAY := move(2)
}
```

result:

```
MO = Dec.
DAY = 07
YR = 1941
```

In each case YR is being assigned the subject string of four characters from the first Len(4) or Move(4) functions. Then after a blank space, MO is assigned the subject string of four characters from the second Len(4) or Move(4) functions. Finally after another blank space, Day is assigned the two letter subject string from the Len(2) or move(2) functions.

3.3.2 Span and Break

Span(c) will match any continuous set of characters in the cset c, Break(c) will match any continuous set of characters upto a character that is a member of cset c. Unicon string scanning accomplishes this with the combination of the tab() and many(c) functions. Some examples are as follows:

SNOBOL4:

```
subjectString = '1941 Dec. 07'
subjectString  SPAN('0123456789') $ year ' ' BREAK(' ') $ month
```

Unicon Patterns (non-anchored mode):

```
pattern := Span(&digits) $$ year && " " && Break(' ') $$ month
subjectString := "1941 Dec. 07"
subjectString ?? pattern
```

Unicon Patterns (anchored mode):

```
subjectString := "1941 Dec. 07"
pattern := Span(&digits)
pattern2 := Break(' ')
subjectString ? {
   year := =pattern
   move(1)
   month := =pattern2
}
```

Unicon Strings:

```
   subjectString := "1941 Dec. 07"
   subjectString ? {
      year := tab(many(&digits))
      move(1)
      month := tab(many(~' '))
   }
```

result:

```
   year = 1941
   month = Dec.
```

In the first two cases the patterns will return the subject string of the first set of digits it encounters, and the last two cases it will the subject string of digits at the beginning of the string, all of which are "1941". After that they pass over the space and the Break() and tab(many()) functions assign "Dec." to month.

3.3.3 Any and NotAny

Any(c) will match a single character in the subject string that is a member of cset c, NotAny(c) will match a single character in the subject string that is not a member of cset c. In Unicon string scanning a similar result can be accomplished with the combination of the tab() and upto(c) functions followed by move(1). Some examples are as follows:

SNOBOL4:

```
   vowels = 'aeiou'
   pattern = (ANY(vowels) NOTANY(vowels)) . result
   subjectString = 'bought'
   subjectString  pattern
```

Unicon Patterns (non-anchored mode):

```
   vowels := 'aeiou'
   pattern := (Any(vowels) && NotAny(vowels)) -> result
   subjectString := "bought"
   subjectString ?? pattern
```

Unicon Patterns (anchored mode):

```
   vowels := 'aeiou'
   pattern := Any(vowels) && NotAny(vowels)
   subjectString := "bought"
   subjectString ? {
      tab(upto(vowels))
      tab(upto(~vowels))
      &pos := &pos - 1
      result := =pattern
   }
```

Unicon String Scanning:

```
   vowels := 'aeiou'
   subjectString := "bought"
   subjectString ? {
      tab(upto(vowels))
      tab(upto(~vowels))
      &pos := &pos - 1
      result := move(2)
   }
```

result:

```
   result = ug
```

In the first two examples the patterns are in non-anchored mode and will scan the subject string until it finds a match of the pattern, being a vowel followed by a non-vowel. In the third example the pattern match is performed in the anchored mode. Without first moving the cursor forward with the string scanning functions, the match will fail as 'b' is not a vowel. In the third and fourth examples some code was added to move the cursor to the location so that the result will match the first two.

3.3.4 Pos and Rpos

Pos(n) and Rpos(n) are primitive pattern functions which set the cursor position. Pos(n) sets the position from the left end of the subject string with the first position being 1 and Rpos(n) sets the cursor position from the right end of the subject string with the last position being 0.

SNOBOL4:

```
   pattern = (POS(3) LEN(4)) . result
   subjectString = 'Unicon Programming'
   subjectString  pattern
```

Unicon Patterns (non-anchored mode):

```
pattern := (Pos(3) && Len(4)) -> result
subjectString := "Unicon Programming"
subjectString ?? pattern
```

Unicon Patterns (anchored mode):

```
pattern := Len(4)
subjectString := "Unicon Programming"
subjectString ? {
   &pos := 3
   result := =pattern
}
```

Unicon String Scanning:

```
subjectString := "Unicon Programming"
subjectString ? {
   &pos := 3
   result := move(4)
}
```

result:

```
result = icon
```

In all four of the examples the operation starts by moving the cursor to the third cursor position from the left of the subject string, then it matches the next four characters of the subject string, resulting in ”icon”.

3.3.5 Tab and Rtab

Tab(n) and Rtab(n) are primitive pattern functions that will match all characters in the subject string from the current cursor position to the nth cursor position. The string scanning function tab(n) has the same functionality, but when there is a negative value for n then it the functionality of Rtab(n).

SNOBOL4:

```
pattern = (POS(3) TAB(7) RTAB(4)) . result
subjectString = 'Unicon Programming'
subjectString  pattern
```

Unicon Patterns (non-anchored mode):

```
pattern := (Pos(3) && Tab(7) && Rtab(4)) -> result
subjectString := "Unicon Programming"
subjectString ?? pattern
```

Unicon Patterns (anchored mode):

```
pattern := Tab(7) && Rtab(4)
subjectString := "Unicon Programming"
subjectString ? {
   &pos := 3
   result := =pattern
}
```

Unicon String Scanning:

```
subjectString := "Unicon Programming"
subjectString ? {
   &pos := 3
   result := tab(7)
   result +:= tab(-4)
}
```

result:

```
result = icon Program
```

In all four of the examples the operation starts by moving the cursor to the third cursor position from the left of the subject string, then it matches the substring upto cursor position 7 resulting in "icon." Then it matches the substring from cursor position 7 to the cursor position which is 4 positions from the right and adding it the previous match, resulting in "icon Program".

3.3.6 Arb and Rem

Arb() is a primitive pattern function that will match an arbitrary number of characters in the subject string bounded by the cursor positions of the patterns on the right and left of the expression. Rem() is a primitive pattern function that will match all the characters of the subject string from the current position to the end of the subject string.

SNOBOL4:

```
pattern = (ANY(' ') ARB  'o') $ result1 REM $ result2
subjectString = 'Unicon Programming'
subjectString  pattern
```

Unicon Patterns (non-anchored mode):

```
pattern := (Any(' ') && Arb() && "o") $$ result1 && Rem() $$ result2
subjectString := "Unicon Programming"
subjectString ?? pattern
```

Unicon Patterns (anchored mode):

```
pattern := (Arb() && "o") $$ result1 && Rem() $$ result2
subjectString := "Unicon Programming"
subjectString ? {
   tab(upto(' '))
   result := =pattern
}
```

Unicon String Scanning:

```
subjectString := "Unicon Programming"
subjectString ? {
   tab(upto(' '))
   result1 := tab(upto('o')) || move(1)
   result2 := tab(0)
}
```

result:

```
result1 =  Pro
result2 = gramming
```

In all four of the examples the operation starts by moving the cursor to the position just before the space. Then result1 is assigned ' Pro' with the Arb() function in the first three examples or tab(upto('o')) —— move(1) set of functions for the fourth example.

3.3.7 Bal

Bal() is a primitive pattern function that will match any non-null string that is balanced with parentheses. The Unicon string scanning bal() has several arguments and returns the cursor position at which the string is balanced in regards to the character in the arguments.

SNOBOL4:

```
pattern = BAL
subjectString = '(3+5)*2'
subjectString  pattern
```

Unicon Patterns (non-anchored mode):

```
pattern := Bal() -> result
subjectString := "(3+5)*2"
subjectString ?? pattern
```

Unicon Patterns (anchored mode):

```
pattern := Bal()
subjectString := "(3+5)*2"
subjectString ? {
   result := =pattern
}
```

result:

```
result = (3+5)
```

In these examples the Bal function locates the beginning parenthesis and then matches the everything until it reaches the closing parenthesis in which the opening and closing parentheses are balanced. If the function begins at a non parentheses then it will return that character as it is a set with a parentheses depth of 0.

3.3.8 Breakx and Arbno

** Arb() is a primitive pattern function that will match an arbitrary number of characters in the subject string bounded by the cursor positions of the patterns on the right and left of the expression. Rem() is a primitive pattern function that will match all the characters of the subject string from the current position to the end of the subject string.

SNOBOL4:

```
pattern = (ANY(' ') ARB  'o') $ result1 REM $ result2
subjectString = 'Unicon Programming'
subjectString  pattern
```

Unicon Patterns (non-anchored mode):

```
pattern := (Any(' ') && Arb() && "o") $$ result1 && Rem() $$ result2
subjectString := "Unicon Programming"
subjectString ?? pattern
```

Unicon Patterns (anchored mode):

```
pattern := (Arb() && "o") $$ result1 && Rem() $$ result2
subjectString := "Unicon Programming"
subjectString ? {
   tab(upto(' '))
   result := =pattern
}
```

Unicon String Scanning:

```
subjectString := "Unicon Programming"
subjectString ? {
   tab(upto(' '))
   result1 := tab(upto('o')) || move(1)
   result2 := tab(0)
}
```

result:

```
result1 =  Pro
result2 = gramming
```

In all four of the examples the operation starts by moving the cursor to the position just before the space. Then result1 is assigned ' Pro' with the Arb() function in the first three examples or tab(upto('o')) —— move(1) set of functions for the fourth example.

3.3.9 Fail, Fence and Cancel

*** Arb() is a primitive pattern function that will match an arbitrary number of characters in the subject string bounded by the cursor positions of the patterns on the right and left of the expression. Rem() is a primitive pattern function that will match all the characters of the subject string from the current position to the end of the subject string.

SNOBOL4:

```
pattern = (ANY(' ') ARB  'o') $ result1 REM $ result2
subjectString = 'Unicon Programming'
subjectString  pattern
```

Unicon Patterns (non-anchored mode):

```
pattern := (Any(' ') && Arb() && "o") $$ result1 && Rem() $$ result2
subjectString := "Unicon Programming"
subjectString ?? pattern
```

Unicon Patterns (anchored mode):

```
pattern := (Arb() && "o") $$ result1 && Rem() $$ result2
subjectString := "Unicon Programming"
subjectString ? {
   tab(upto(' '))
   result := =pattern
}
```

Unicon String Scanning:

```
subjectString := "Unicon Programming"
subjectString ? {
   tab(upto(' '))
   result1 := tab(upto('o')) || move(1)
   result2 := tab(0)
}
```

result:

```
result1 =  Pro
result2 = gramming
```

In all four of the examples the operation starts by moving the cursor to the position just before the space. Then result1 is assigned ' Pro' with the Arb() function in the first three examples or tab(upto('o')) —— move(1) set of functions for the fourth example.

4 Implementation

References

[1] Sudarshan Gaikaiwari. *Adapting SNOBOL-style Patterns to the Unicon Language*. PhD thesis, New Mexico State University, 2005.

[2] Ralph E Griswold. *Pattern Matching in Icon*. University of Arizona, Department of Computer Science, 1980.

[3] Ralph E Griswold and Madge T Griswold. *The Icon programming language*, volume 30. Prentice-Hall Englewood Cliffs, NJ, 1983.

[4] Ralph E Griswold, Ivan Paul Polonsky, and JF Poage. The snobol4 programming language. 1971.

[5] Clinton Jeffery, Shamim Mohamed, Ray Pereda, and Robert Parlett. Programming with unicon. *Draft manuscript from http://unicon. sourceforge. net*, 2004.