Integrating pattern data types within Unicon string scanning

John H. Goettsche

Dept. of Computer Science

University of Idaho

August 5, 2014

Abstract

The SNOBOL4 pattern data type and the pattern matching process were introduced to the Unicon language in 2005, but was not fully integrated with the Unicon string scanning environment. The goal of this project is to make modest changes to the pattern matching functions and make the pattern data type accessible to the Unicon string scanning environment. To accomplish this, a Unicon string scanning operator was changed to to execute a pattern match in the anchored mode in order to preserve the logic behind the string scanning operation.

Contents

1 Introduction **4**

2 Background **5**

3 Design Considerations **6**

3.1 Pattern matching statements 7

3.1.1 Patterns into String Scanning 8

3.1.2 String Scanning into Patterns 9

3.2 SNOBOL4 and Unicon pattern operators 9

3.3 SNOBOL4 and Unicon pattern functions 10

3.3.1 Logically Redundant Function 11

3.3.2 Index Related Functions 11

3.3.3 Backtracking and Terminating Functions 12

4 Implementation **13**

4.1 Pattern matching statements 13

4.1.1 Anchored mode from string scanning 14

4.1.2 Anchored mode resulting from string scanning functions . 16

4.2 Revised Pattern Functions 16

4.2.1 Logically Redundant Function 17

4.2.2 Index Related Functions 17

5 Evaluation **19**

6 Conclusions **19**

7 Future Work **19**

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 requlars 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. [6] 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

SNOBOL was developed by David Farber, Ralph Griswald and Ivan Polonsky at Bell Telephone Laboratories in 1962. SNOBOL4 was developed in 1967 and had many of the features that are included in popular dynamic programming languages including dynamic typing, eval and garbage collection. Its pattern data type was its most important contribution to string processing. 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. [6] Icon examined SNOBOL4's generators and backtracking in its pattern matching to develop and implement goal directed evaluation. [1] SNOBOL4 patterns were not incorporated in Icon, instead Griswald developed an extensive string scanning system where a veritey of functions and operations are executed in order to analyze and manipulate a subject string as a

cursor advances through that string. Unlike most other languages Icon considers the string as a data type in its own right, rather that a set of characters. [3]

Using the same Icon source code, Unicon was developed to include modern software features such as objects, networks and databases. [6] The string scanning environment of Icon is a part of the Unicon programming language. Master's student, Sudarshan Gaikaiwari proposed adopting SNOBOL patterns to the Unicon language. In his thesis, he added the pattern data type, and provided pattern matching functions and operators to execute the pattern searches. [1] The pattern data type was kept separate from the string scanning environment which may not execute pattern matching operations. These pattern matching operations are not integrated into 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

To integrate pattern matching with Unicon string string scanning we have to consider how we would implement patterns in the string scanning environment and string scanning in the pattern matching operation. The pattern functions and operators will have to be consistent lexically and functionally with the Unicon language.

3.1 Pattern matching statements

SNOBOL pattern matching can be executed in either anchored or non-anchored mode. The anchored mode requires the match to start on the first character of the subject string while in the non-anchored mode the match can start at any location in the subject string. [4] The pattern matching operation adapted by Gaikaiwari are generators. [1] They only produce one value, but since they suspend instead of return that value, the cursor position is stored. This allows the the operation to produce the next value. [6]

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. In the anchored mode of the SNOBOL4 operation the pattern has to begin its match at the first character of the subject string; 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]

To integrate pattern matching to the Unicon language we have to consider which mode is appropriate considering which environment it is in. If it is being

executed outside the Unicon string scanning environment, the cursor position or index of the string has not been established. Therefore the non-anchored mode would be appropriate, but it the user wishes to have the pattern match run in the anchored mode, starting the pattern with Pos(1) would have the desired effect.

3.1.1 Patterns into String Scanning

Executing pattern matching from within the Unicon string scanning environment. The &pos is established and string scanning environment uses an inductive method of executing a series of expressions to manipulate and analyze a subject string. The cursor location or index is adjusted depending on the success or failure of each expression. In order to maintain this systematic process in the string scanning environment, it was decided to execute the pattern matching operation in the anchored mode.

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.

3.1.2 String Scanning into Patterns

Given that string scanning functions require an index location, using them in the pattern matching operation will require the cursor location or index to be established prior to their execution. Considering that the pattern matching operation in the non-anchored mode does not start from a fixed index location, it is established when a pattern is found. Therefore, it will be necessary to set the mode of the pattern match to anchored and an index location be established when a pattern contains a string scanning function.

3.2 SNOBOL4 and Unicon pattern operators

The pattern operators for Unicon were defined by Gaikaiwari in his Master's Thesis. Although they are lexically different, they are functionally identical to SNOBOL4 pattern operators. Gaikaiwari's operators for concatenation and alternation do not match either the Unicon operators. The Unicon operators for concatenation and alternation should be modified to recognize whether the operator is a pattern or a standard operator.

Table 1: Pattern Operators

Operation	SNOBOL4	Gaikaiwari
Concatenation	<<implicit>>	&&
Alternation	|	.|
Immediate Assignment	$	$$
Conditional Assignment	.	$-$ >
Cursor Assignment	@	.$
Unevaluated Expression	*x	‘x‘

3.3 SNOBOL4 and Unicon pattern functions

In the Unicon string scanning environment the functions operate in relation to the cursor position of the subject string. It is a inductive process where the data is analyzed by injecting the functions into the subject string. While the patterns are pre-defined and are used deductively to search a subject string for the pattern. In the pattern matching operation many of the functions operate independently of the cursor location and later determining a new cursor location to as a part of determining the results of the pattern match. This difference lead the author to conclude that the SNOBOL pattern function names should be retained as much as possible while altering to fit a Unicon naming conventions.

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. Other changes are described in the example uses of the functions bellow:

Table 2: Pattern Functions

SNOBOL4	Proposed
LEN(n)	Len(n)
SPAN(c)	Span(c)
BREAK(c)	Break(c)
ANY(c)	Any(c)
NOTANY(c)	NotAny(c)*
TAB(n)	Tab(n)
RTAB(n)	Rtab(n)*
REM	Rem()
POS(n)	Pos(n)
RPOS(n)	Rpos(n)*
FAIL	Back()*
FENCE	Fence()
ABORT	Cancel()*
ARB	Arb()
ARBNO(p)	Arbno(p)
BAL	Bal()

* described in subsection below

3.3.1 Logically Redundant Function

The complement operator allows the user to get the complement of a given cset, or a cset containing all the characters not included in the given cset. Therefore the functionality of NotAny(c) function can be achieved by using the complement operator with a cset in the Any function as Any(c). The run time for each method is the same on average.

3.3.2 Index Related Functions

Pos, Rpos, Tab and Rtab functions all work directly with the cursor location or index. In Unicon the index value is the number of spaces to the right from the left end of string with the first position being 1 or the number of spaces subtracted

from the right end of the string. [6] The illustration below demonstrates the cursor position values for the string "Unicon" with the vertical bars representing the index locations:

```
-6  -5  -4  -3  -2  -1  0
| U | n | i | c | o | n |
1   2   3   4   5   6   7
```

With the Rpos and Rtab functions the cursor locations are as follows:

```
6   5   4   3   2   1   0
| U | n | i | c | o | n |
```

3.3.3 Backtracking and Terminating Functions

Fail is a already taken as a keyword in Unicon. Attempts to use this name created a lot problems in compiling Unicon. Fail in Unicon means that there is not a successful result in the operation. While performing the pattern matching operation, FAIL is used to signify that there is not a successful result in the current pattern element and instructs the system to backtrack and to try another alternative. Since the function is localized to a pattern match element and is not intended for a failure of the entire operation, I proposed to use Back() for the SNOBOL4 FAIL function.

Likewise ABORT is a SNOBOL function that cancels the pattern matching operation, but does not the operation entire program. Cancel would be a more appropriate term for the ABORT function.

4 Implementation

To implement these changes the following changes to the Unicon language had to be made:

- Modify Unicon's runtime to execute pattern matching in the anchored mode when it is executed in the tabmat operator in the string scanning environment.
- Modify the pattern matching operation to recognize string scanning functions.
- Add pattern scan to identify its contents and determine the pattern matching mode.
- Modify the pattern source files to address the functional changes for Pos and Tab.
- Modify function definitions for the Unicon build.
- Modify the concatenation and alternation operators to function with patterns.

4.1 Pattern matching statements

The non-anchored pattern matching operation was resolved by Sudarshan Gaikaiwari in his 2005 Master's theses at New Mexico State University. A non-anchored pattern matching expression consists of a subject followed by the comparison operator '??' followed by a pattern and appears as follows:

```
subject ?? pattern
```

The anchored pattern matching operation was defined in the pattern resources as a part of the internal match function, but had the Anchored_Mode identifier set to false. The default location of the index was set to 1. The arguments for the internal match was changed so that the mode would be passed in along with the current index. This allows the internal match to be used called and initiated in the proper location of the subject string.

4.1.1 Anchored mode from string scanning

It was decided that integrating the pattern matching system into the Unicon string scanning environment would require the pattern matching be performed in the anchored mode, since the string scanning functions are dependent upon the index or cursor position. For this implementation the Unicon tabmat operator '=' was determined to be an ideal choice for initiating a pattern match in the string scanning environment. The use of an equals '=' before a pattern variable triggers the anchored mode pattern matching operation.

```
subjectString ? {
   match := =pattern
}
```

The tabmat operator was modified to accept pattern data types. In the event that a pattern was its argument then it would have to initiate a pattern match in the anchored mode, otherwise it would have to function normally. This section of code identifies if the argument is a pattern or a string and assigns its return value.

```
operator{*} = tabmat(x)
   declare {
      int use_trap = 0;
```

```
   }
/*
 * x must be a pattern or convertible into a string.
 */
   if is:pattern(x) then {
      inline {
         use_trap = 1;
      }
      abstract {
         return string
      }
   } else if !cnv:string(x) then {
      runerr(103, x)
   } else
      abstract {
         return string
      }
```

For the pattern match, the body of code had to assign the values for each of the variables required in the pattern match. The index or cursor location had to be assigned so that the anchored pattern match would begin where the string scanning had left off. Finally, if the pattern was successful, then it would have to suspend the matching pattern update the index or cursor locations, if it failed then it would revert to the previous index. The following code was added to the body of the tabmat operator:

```
/*
 * set cursor position, and subject to match
 */
oldpos = curpos = k_pos;
pattern_subject = StrLoc(k_subject);
subject_len = StrLen(k_subject);
pattern = (struct b_pattern *)BlkLoc(x);
phead = ResolvePattern(pattern);
/*
 * runs a pattern match in the Anchored Mode and returns
 * a sub-string if it succeeds.
 */
if (internal_match(pattern_subject, subject_len, pattern->stck_size,
```

```
      phead, &start, &stop, curpos - 1, 1)){
   /*
    * Set new &pos.
    */
   k_pos = stop + 1;
   EVVal(k_pos, E_Spos);
   oldpos = curpos;
   curpos = k_pos;
   /*
    * Suspend sub-string that matches pattern.
    */
   suspend string(stop - start, StrLoc(k_subject)+ start);
   pattern_subject = StrLoc(k_subject);
   if (subject_len != StrLen(k_subject)) {
      curpos += StrLen(k_subject) - subject_len;
      subject_len = StrLen(k_subject);
   }
}
/*
 * If tab is resumed, restore the old position and fail.
 */
printf("oldpos: %d, StrLen: %d\n", oldpos, StrLen(k_subject) + 1);
if (oldpos > StrLen(k_subject) + 1){
   runerr(205, kywd_pos);
} else {
   k_pos = oldpos;
   EVVal(k_pos, E_Spos);
}
```

4.1.2 Anchored mode resulting from string scanning functions

This is still under review.

4.2 Revised Pattern Functions

By revising the function definitions file, 'fdef.h', and the patterns source file 'fx-pattrn.ri' in Unicon source files. The pattern function names have been revised to be lexically more consistent with the Unicon language as shown on the table bellow:

Table 3: Pattern Functions

SNOBOL4	Proposed
LEN(n)	Len(n)
SPAN(c)	Span(c)
BREAK(c)	Break(c)
ANY(c)	Any(c)
NOTANY(c)	included in Any(c)
TAB(n)	Tab(n)
RTAB(n)	included in Tab(n)
REM	Rem()
POS(n)	Pos(n)
RPOS(n)	included in Pos(n)
FAIL	Back()
FENCE	Fence()
ABORT	Cancel()
ARB	Arb()
ARBNO(p)	Arbno(p)
BAL	Bal()

Rtab(n) and Rpos(n) are redundant functions that can be performed using Tab(n) and Pos(n) with the Unicon string index system. NotAny(c) can be handled just as well with the complement operator with the cset in the Any(c) function.

4.2.1 Logically Redundant Function

Although the complement operator allows the user to get the complement of a given cset and it can be used with a cset in the Any(c) function, NotAny(c) is still and option for the user. It is a redundant function which may be desirable for those who have extensive experience with SNOBOL. The run time for both methods are the same on average.

4.2.2 Index Related Functions

The pattern cursor location representation in the pattern functions have been revised to match the Unicon index location of strings in the following form.

```
-6  -5  -4  -3  -2  -1  0
| U | n | i | c | o | n |
1   2   3   4   5   6   7
```

This was achieved by changing the Pos(n) function code to appear as follows:

```
function {1} Pos(position)
   abstract {
      return pattern;
   }
   body {
      union block *bp;
      /*
       * check if position is negative
       */
      if(position.vword.integr < 1) {
         /* change position to a positive value and use RPos */
         position.vword.integr = -position.vword.integr;
         ConvertPatternArgumentInt(position,bp,PC_RPos);
      } else {
         ConvertPatternArgumentInt(position,bp,PC_Pos);
      }
      return pattern(bp);
      }
end
```

The position that is passed to the function is a descrip object in the Unicon virtual machine that is defined to be an integer. The details of the descrip object can be found in Implementation of Icon and Unicon. [5] The if statement in this function check to see if the value of the descrip object is negative. If it is negative then its value is changed to its complement the the ConvertPatternArgumentInt function is called while passing position, a block pattern and the PC_RPos call as arguments. If it is positive then position is not changed and the PC_Pos argument is used instead while calling the ConvertPatternArgumentInt function. A nearly identical changes ware made to Tab function.

The Rpos(n) and Rtab functions are now redundant, but are still available for the user who may be more comfortable with SNOBOL. When using Rpos(n) and Rtab(n), the user has to be aware that they are bosed on the pattern matching cursor location system as follows:

```
6   5   4   3   2   1   0
| U | n | i | c | o | n |
```

5 Evaluation

This implementation of SNOBOL4 patterns in the Unicon language is consistent with the lexical and functional structure of Unicon. It also integrates the pattern matching system into Unicon's he string scanning environment.

6 Conclusions

The implementation of patterns within the string scanning environment was successful. A unicon programmer is now able to use pattern matching in the string scanning environment in the anchored mode.

7 Future Work

For the pattern data type to be fully integrated into the Unicon Language, the string scanning functions need to be integrated into the pattern data type and pattern matching operation.

Also the variables in patterns should be unevaluated by default.

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. Implementation of icon and unicon.

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