Integrating pattern matching within string scanning

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

January 10, 2015

Abstract

A SNOBOL4 like pattern data type and pattern matching operation were introduced to the Unicon language in 2005, but patterns were not integrated with the Unicon string scanning control structure at that time. The goal of this project is to make the pattern data type accessible to the Unicon string scanning and vice versa. To accomplish this, a Unicon string scanning operator was changed to allow the execution of a pattern match in the anchored mode.

Contents

1 Introduction

In order to enhance programmers' productivity in analyzing strings, string scanning and pattern matching systems have been developed. Modern high-level languages tend to offer regular expressions and context free grammars for their string processing. Analysis of strings often requires functionality beyond what these classes of languages offer. SNOBOL4 was the most successful of the early string processing languages. [1, 2] A pattern data type was employed in SNOBOL4 along with pattern operators and functions to perform the analysis.

A successor to SNOBOL4 was Icon, which expanded upon the goal directed evaluation with generators and backtracking that was implicit in SNOBOL4's pattern matching. [2, 3] It uses a string scanning method which passes through the subject string with a set of commands to manipulate and analyze its contents. Many of the Unicon string scanning fucntions resemble SNOBOL 4 patterns, but they are different in both how they are processed as well as their functionality. Many Icon programmers have expressed a desire for the functionality of SNOBOL4 patterns. [4].

When Unicon was developed, its core elements came directly from Icon, where it inherited its string scanning functions. [5] While patterns are instances of a structured data type that are used deductively to test whether the pattern exists within a subject string, the string scanning environment uses a set of functions to inductively analyze or extract data from the subject string. Patterns are pre-defined and applied later, while string scanning is performed as string scanning functions when are executed. Patterns allow composition and re-use in more flexible ways than is the case for code, which only has procedure and co-expression granularity.

This paper briefly explores the background of SNOBOL4 patterns and Unicon string scanning functions. It then presents a proposed design for integrating SNOBOL4 patterns within Unicon string scanning environments, and a description of how that implementation can be accomplished.

2 Background

SNOBOL was developed by David Farber, Ralph Griswold 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 they could be a complex arrangement with alternative character sets and pattern functions. The pattern data type enabled the user to define and store patterns as variables to be used later when they were desired. [1]

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] Griswold used his experience with SNOBOL4's generators and backtracking in its pattern matching to develop and implement goal directed evaluation. [2] SNOBOL4 patterns were not incorporated in Icon. Instead Griswold developed an extensive string scanning system where a variety 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 than an array of characters. [3]

Using the same Icon source code, Unicon was developed to include modern software features such as objects, networks and databases. [5] The string scanning control structure of Icon is a part of the Unicon programming language. Sudarshan Gaikaiwari adopted SNOBOL patterns to the Unicon language for his master's thesis. In his thesis, he added the pattern data type, and provided pattern matching functions and operators to execute the pattern searches. [2] The pattern data type was not integrated and kept separate from the string scanning environment which may not execute pattern matching operations. This paper proposes to refine Gaikaiwari's work to be more naturally incorporated in the Unicon language allowing the string scanning environment to implement the pattern matching functionality.

3 Design Considerations

To integrate pattern matching with Unicon string string scanning requirements, consideration of how to implement patterns in the string scanning environment and how to implement 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. [1] The pattern matching operation adapted by Gaikaiwari are generators and operate in the non-anchored mode. [2] They only produce one value at a time, but since they suspend instead of return that value, the cursor position is stored so the next pattern can be applied to the remainder of the string to produce the next value. [5]

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

SNOBOL4	Gaikaiwari's Unicon
`SUBJECT PATTERN`	`subject ??  pattern`

In both examples, the subject is scanned to see if it contains the pattern. If it succeeds, then a substring of the subject that fits the pattern is produced. Gaikaiwari's syntax starts the pattern matching operation with the use of the `??` operator while SNOBOL4 uses two spaces between the subject and pattern. Gaikaiwari's is easier to read and is a better match to Unicon syntactically.

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 of a 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 if 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 Pattern matching within String Scanning

When 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 expression:

```
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. Scanning functions normally move the cursor upon success, for this reason, I propose 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. 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 are lexically different from Unicon concatenation and alternation and the assignment operators are lexically inconsistent.

Table 1: Pattern Operators

Operation	SNOBOL4	Gaikaiwari	Proposed
Concatenation	<<implicit>>	&&	||
Alternation	|	.|	.|
Immediate Assignment	$	$$	=>
Conditional Assignment	.	− >	− >
Cursor Assignment	@	.$	.>
Unevaluated Expression	∗x	‘x‘	‘x‘

The Unicon operator for concatenation will be modified to recognize whether the expression contains a pattern.

The assignment operators will be changed lexically to use the > symbol to represent an assignment within patterns. The Immediate Assignment, Conditional Assignment and Cursor assignment will be =>, − > and .> respectively.

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 an 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 determine a new cursor location as a part of determining the results of the pattern match. This difference led the author to conclude that the SNOBOL pattern function names should be retained as much as possible while altering to fit 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 exceptions for FAIL and ABORT. Other changes are described in the example uses of the functions 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

Table 2: Pattern Functions

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

* see logically redundant functions described in subsection 3.3.1 below

** see index related functions described in subsection 3.3.2 below

*** see backtracking and terminating functions in subsection 3.3.3 below

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(n)`, `RPOS(n)`, `TAB(n)` and `RTAB(n)` SNOBOL 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. [5] 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
```

The SNOBOL RPOS(n) and RTAB(n) functions the cursor locations are as follows:

```
RTAB(n)    6   5   4   3   2   1   0
           | S | N | O | B | O | L |
TAB(n)     1   2   3   4   5   6   7
```

Integration of the SNOBOL RPOS(n) and RTAB(n) functions with the Unicon string indexes can be achieved with Pos(n) and Tab(n) making RPOS(n) and RTAB(n) redundant.

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, it makes sense to use Back() for the SNOBOL FAIL function.

Likewise ABORT is a SNOBOL function that cancels the pattern matching operation, but does not halt the operation of the 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 implemented by Sudarshan Gaikaiwari in his 2005 Master's thesis 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 were changed so that the mode would be passed in along with the current index. This allows the internal match to be 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 initiates a pattern match in the anchored mode, otherwise it functions 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.
      */
     if (oldpos > StrLen(k_subject) + 1){
        runerr(205, kywd_pos);
     } else {
        k_pos = oldpos;
        EVVal(k_pos, E_Spos);
     }
```

4.1.2 String and Pattern Concatenation Operators

It was decided to have a consistent concatenation operator, therefore the pattern concatenation operator && had to be removed and the string concatenation operator || would have to be used in its place. The existing string concatenation operator accepted only strings. Pattern concatenation accepts strings, c-sets and patterns. Therefor the string concatenation operator had to changed to accept c-sets and patterns. String concatenation and pattern concatenation are handled in separate operations and are performed on their respective data types; strings must be sent to the string concatenation operator and patterns must be sent to the pattern concatenation operator.

```
   declare {
      int use_trap = 0;
      }

   if is:pattern(x) then {
      inline {
         use_trap = 1;
         }
      abstract {
         return pattern;
         }
      }
   else if is:pattern(y) then {
      inline {
         use_trap = 1;
         }
      abstract {
         return pattern;
         }
      }
   else if is:cset(x) then {
      inline {
         use_trap = 1;
         }
      abstract {
         return pattern;
         }
      }
   else if is:cset(y) then {
      inline {
         use_trap = 1;
         }
      abstract {
         return pattern;
         }
      }
   else {
      if !cnv:string(x) then
      runerr(103, x)
      if !cnv:string(y) then
      runerr(103, y)
      }

if (use_trap == 1) {
union block *bp;
struct b_pattern *lp;
struct b_pattern *rp;
struct b_pelem *pe;
lp = (struct b_pattern *)BlkLoc(x);
rp = (struct b_pattern *)BlkLoc(y);
if(is:string(x))cnv_str_pattern(&x, &x);
if(is:string(y))cnv_str_pattern(&y, &y);
pe = Concat(Copy((struct b_pelem *)lp->pe), Copy((struct b_pelem *)rp->pe), rp->stck_size);
bp = pattern_make_pelem(lp->stck_size + rp->stck_size,pe);
return pattern(bp);
}else {
```

4.2 Revised Pattern Functions

By revising the function definitions file, `fdef.h`, and the patterns source file `fxpattrn.ri` in Unicon source files. The pattern function names have been revised to be lexically more consistent with the Unicon language. the table below condenses the Table 2 to show the SNOBOL4 pattern functions with their corresponding Unicon functions for this implementation:

Table 3: Pattern Functions

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

4.2.1 Index Related Functions

The pattern cursor location representation in the pattern functions have been revised to match the Unicon index location of strings as shown in section 3.3.2 of this paper. 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 `struct descrip` in the Unicon virtual machine that is defined to be an integer. The details of the `struct descrip` can be found in Implementation of Icon and Unicon. [6] The `if` statement in this function checks 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 based on the SNOBOL4 pattern matching cursor location system.

5 Evaluation

To evaluate the how successful this implementation of SNOBOL4 type patterns in Unicon, example code to execute a benchmark problem will be compared with SNOBOL4 and Gaikaiwari's pattern statements, this proposal's pattern statements from both the anchored string scanning environment and the non-anchored pattern matching environment, and equivalent Unicon string scanning functions. They will be checked for clarity, simplicity and functionality. Clarity deals with the readability of the lines of code for each particular problem. Can the user or programmer read the code without any ambiguity to what it is attempting to achieve. Simplicity will be measured by the number of lines required to achieve each benchmark problem. Functionality of the example code will have to be consistent. Each example will have to achieve the same result for each benchmark problem. The benchmark problems used in Gaikaiwari's thesis were:

- Decomposing phone numbers
- Detecting words with double letters
- Strings of the form $A^nB^nC^n$

5.1 Decomposing phone numbers

For the purpose of Decomposing phone numbers in North America, each piece of example code will have to be able to identify the area code, trunk and the remainder of the number. The following example phone numbers should be recognizable:

1. 800-555-1212
2. 800 555 1212
3. 800.555.1212
4. (800) 555-1212
5. 1-800-555-1212
6. 1-(800) 555-1212
7. 800-555-1212-1234
8. 800-555-1212 ext. 1234

An input fragment of `Home: (800) 555-1212 ext. 1234` should result in identifying the area code as `800`, the trunk as `555` and the remainder of the number as being `1234`.

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] R. E. Griswold, I. P. Polonsky, and J. Poage, *The SNOBOL4 Programming Language*. Prentice-Hall, 1971.

[2] S. Gaikaiwari, "Adapting SNOBOL-style Patterns to the Unicon Language," Master's thesis, New Mexico State University, 2005.

[3] R. E. Griswold and M. T. Griswold, *The Icon Programming Language*. Prentice-Hall Englewood Cliffs, NJ, 3 ed., 1983.

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

[5] C. Jeffery, S. Mohamed, R. Pereda, and R. Parlett, *Programming with Unicon*. 2004.

[6] C. Jeffery, *Implementation of Icon and Unicon*.