Integrating pattern data types within Unicon string scanning

John H. Goettsche

Dept. of Computer Science

University of Idaho

July 30, 2014

Abstract

The SNOBOL4 pattern data type and the pattern matching process were to the Unicon language in 2005, but was not fully integrated with the Unicon string scanning environment. Modest changes were made to the pattern matching functions and the pattern data type was made accessible to the Unicon string scanning environment. A Unicon string scanning operator was changed to to execute a pattern match in the anchored mode in order to preserve the logic of the string scanning operation.

Contents

1 Introduction **3**

2 Background **4**

3 Design Considerations **5**

3.1 Pattern matching statements 6

3.2 SNOBOL4 and Unicon pattern operators 7

3.3 SNOBOL4 and Unicon pattern functions 8

3.3.1 Pos, Rpos, Tab and Rtab 9

3.3.2 Back and Cancel 10

4 Implementation **10**

4.1 Pattern matching statements 10

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. [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

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. [5] 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. [5] 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

There are several questions that had to be resolved on how patterns are to used in a Unicon program. In what mode will a pattern matching statement be executed inside or outside the Unicon string scanning environment? How the pattern operators and functions should appear in order to remain consistent with Unicon while being easily recognizable by those who are familiar with SNOBOL? Can the pattern functions be executed in the same manner as similar

string scanning functions and do they have similar functionality?

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 question for integrating SNOBOL's pattern matching to the Unicon language was a question of which mode was appropriate when.

Unicon string scanning environment uses an inductive method of executing a series of expressions to manipulate and analyze a subject string. In this process the cursor location within the subject string is monitored and 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. As for the non-anchored mode, its operation would remain as Gaikaiwari adapted patterns to the Unicon language, where it can be executed outside the string scanning environment.

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

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()

3.3.1 Pos, Rpos, Tab and Rtab

Pos, Rpos, Tab and Rtab functions all work directly with the cursor location. In Unicon the cursor location 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. The illustration below demonstrates the cursor position values for the string "Unicon" with the vertical bars representing the cursor 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 |
```

3.3.2 Back and Cancel

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 no result in the operation, while with pattern matching it is used to signify that there is no result in the current position in the evaluation and to backtrack to try another alternative. Since the function is localized to the pattern match operation and is not intended for a failure of the entire operation, I proposed to use Back() for the SNOBOL4 FAIL 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 function definitions for the Unicon build.
- Modify the pattern source files to address the functional changes for Pos and Tab.

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, but not implemented. It was decided that integrating the pattern matching system into the Unicon string scanning environment. Since the analysis of a string was in progress and the string scanning environment studiously tracks the cursor position as it progresses, it was decided that pattern matching operation would be executed in the anchored mode.

The Unicon tabmat operator '=' was determined to be an ideal choice. The use of an equals '=' before a pattern variable triggers the anchored mode pattern matching operation, as shown in the previous examples in Section 3.3.

The tabmat operator was modified to accept pattern data types. The argument data type test section was modified as follows:

```
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
      }
```

The value for the variable 'use_trap' is set to identify if the argument is a pattern. In the body it is determined which block of code to use. The pattern block was derived from the pattern match function that is in the fxpattrn.ri file.

```
   body {
if (use_trap == 1) {
int curpos;
int oldpos;
int start;
int stop;
struct b_pattern *pattern;
tended struct b_pelem *phead;

char * pattern_subject;
int subject_len;
int new_len;
CURTSTATE();

/*
 * 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);
}
} else {
...
```

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.