

Integrating pattern data types within Unicon string scanning

John H. Goettsche

Dept. of Computer Science

University of Idaho

July 31, 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
3.1.1	Patterns into String Scanning	7
3.1.2	String Scanning into Patterns	8
3.2	SNOBOL4 and Unicon pattern operators	8
3.3	SNOBOL4 and Unicon pattern functions	9
3.3.1	Index Related Functions	10
3.3.2	Backtracking and Terminating Functions	11
4	Implementation	11
4.1	Pattern matching statements	12

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 regular 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 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. [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 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 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 Griswold 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 than 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 Gaikawai 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 Gaikawai 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 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. [5]

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 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 Patterns into String Scanning

Executing pattern matching from within the Unicon string scanning environment. The cursor 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 environment 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 Gaikawai in his Master's Thesis. Although they are lexically different, they are functionally identical to SNOBOL4 pattern operators. Gaikawai'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	Gaikawai
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 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 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 below:

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 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. [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

```

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 Backtracking and Terminating Functions

Fail is already taken as a keyword in Unicon. Attempts to use this name created a lot of 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 match-

ing 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 Gaikawari 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
```

4.1.1 Anchored mode from string scanning

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 tracks the cursor or index 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.

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 fxpatrn.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 Gaikawai. *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.