Integrating pattern data types within Unicon string scanning

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

July 28, 2014

Abstract

1 Introduction

Unicon string scanning functions were inherited from the Icon language. Many programmers have expressed a desire for the functionality of SNOBOL4 patterns. [2] 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 used deductively to test whether the pattern exists within a subject string, while a string scanning uses a set of functions used inductively to analyze or extract data from the subject string. The difference being that patterns are pre-defined and applied, 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 recomended 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	Unicon
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	PFail()
FENCE	Fence()
ABORT	Cancel()
ARB	Arb()
ARBNO(p)	Arbno(p)
BAL	Bal()

Len(n) function matches a string of characters of n length beginning from the current cursor position, for example:

SNOBOL4:

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

Unicon pattern matching:

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

or

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

Unicon string scanning:

```
line := "1941 Dec. 07"
line ? {
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 substring of four characters from the first Len(4) or Move(4) functions. Then after a blank space, MO is assigned the substring of four characters from the second Len(4) or Move(4) functions. Finally after another blank space, Day is assigned the two letter substring from the Len(2) or move(2) functions.

Span(c) will match any continuous set of characters in the cset c, for example:

SNOBOL4:

```
line = '1941 Dec. 07'
line  SPAN('0123456789') . year
```

Unicon Patterns:

```
line := "1941 Dec. 07"
line ?? Span(&digits) -> year
```

or

```
line := "1941 Dec. 07"
pattern := Span(&digits)
line ? {
year := pattern
}
```

Unicon Strings:

```
line := "1941 Dec. 07"
line ? year := tab(many(&digits))
```

result:

```
year = 1941
```

In the first two cases the patterns will return the substring of the first set of digits it encounters, and the last two cases it will return the substring of digits at the beginning of the string, all of which are "1941".

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.