An introduction to KP Language

Tae Yano
tae.yano@hunter.cuny.edu
ty2142@columbia.edu

Sep/27/2004

CSMS W4115
Programming Language and Translator
Fall 2004

Document Type: language overview -- first draft

1. Introduction

The KP Language (this name is tentative) is developed to help knitters of all level of computer literacy to write knitting patterns, or applications to generate knitting patterns, with ease. Using KP, knitters can create new patterns, combine or adjust existing ones for their liking. Any pattern written in knit is by itself an abstraction of a knittable item. The accompanied KP compiler takes source codes written in KP and generates objects defined in Java. Typically, when executed, those objects print out, "blue prints", and/or row-by-row instructions in commonly accepted knitter's notation. Any KP object can actually be made with (at least two) needles and yearn.

2. Feature: What it is and is not

KP is a tool to serve one narrowly defined purpose. It is not a language to write general, all-mighty CAD application or such. Its aim is to do one kind of job and do it well.

The focuses of the language design here are more on simplicity, flexibility, and intuitiveness, rather than versatility. For example, the language, as is, does not enforce strong semantic regiment (thus it is possible to produce ugly, strange, witless knitting patterns). But labor is devoted to build sound/robust body of grammar.

The KP Language is also highly OOP conscious and, as such, highly modular. This, however, is not a goal itself. It is actually the matter of convenience. OOP happen to be a suitable method to abstract the subject at hand (e.g., knitted items). OOP also offers solutions to some problems in the conventional knitters notations (manipulation of objects, reusability of object etc), which is the more ambitious goal of KP language.

3.Background: Knitter's Language

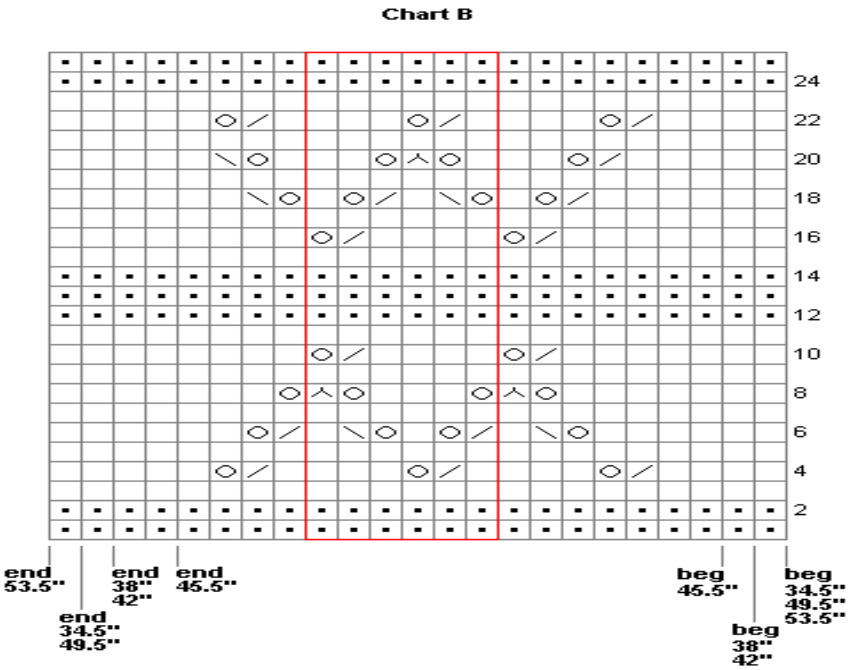
Knitters use a well defined and widely known notation to describe knitted items. There are some dialect, and some creative vernaculars seen occasionally, but the core part of the notation is invariable almost everywhere.

This is how the notaion looks like:

```
ssk:  Sl2 sts knitways one at a time then insert left-hand
       needle into fronts of these 2 sts from the left and knit them
       together.
PATTERN:
       1st Row: K1, * k1, k2together, yfwd, k1, yfwd, ssk, k2; rep from
       * to end.
       2nd and every alt Row: Purl.
       3rd Row: K1, * k2together, yfwd, k3, yfwd, ssk, k1; rep from *
       to end.
       5th Row: K2together, * yfwd, k5, yfwd, s1, k2together, psso; rep
       from * end last rep ssk.
       7th Row: K1, * yfwd, ssk, k3, k2together, yfwd, k1; rep from *
       to end.
Rep these 12 rows.
```

This row-by-row type description is the only reference you need for actual knitting, however, very few people can imagine the finished product from this. For this end, pictograms using grits are often accompanied with written instructions, though rarely no one use pictogram alone when knitting.

This is how such a picture looks like (the pattern is different):



4.Observation : Strength of knitter's language

KP language borrows heavily from the common knitters' notation. The targeted audience (knitters, knitting application designers) are already familiar with those symbols; therefore they can learn the language with relative ease. Moreover, knitter's notation, despite (or because?) it is spontaneously developed over the course of the long time, is surprisingly reasonable, understandable, and intuitive - that is, agreeable to many. It manages to be efficient / succinct without being tares.

Not only it is humanity friendly, the common knitter's notation is linguistically sound/robust. It is capable to describe any items that can be knitted. I have not found out if it is NOT capable to describe what can not be knit, but it is a promising notion.

In fact, the first challenge in KP language development is not "how" to define the language, but "how not" to spoil the existing notation and its strength while formalizing it into "language".

5. Another Observation : Problem of knitter's language

The knitter's notation, of course, is not without a problem.

While the notation is very powerful in describing patterns, it does not have a built-in mechanism to generate new ones. (let me explain)

Previously in this document, two styles of knit pattern description were shown --- a row-by-row instruction written in knitter's notation and a pictogram. the later, while not really useful as a knitting instruction, is good for visualizing the item. knitters often use it when they start new projects or modify the existing ones.

Often, knitters wish to scale up/down, interpose, or combine given patterns. They first draw a pictogram, or convert the existing instruction to one and modify it. They then convert the picture back to the row-by-row instruction when it is ready for knitting. This process, unless in the case of the simplest patterns, almost always involves tedious circulations and lengthy trial-and-error. For the most complicated items, the given patterns are almost un-alterable by hand; even small modifications require rewriting the pattern over.

The source of the problem is that, while knit production (by hands, machine) needs to crunch information sequentially (row-by-row), human's mind does not perceive a knitted item as a stack of strings. It is in their mind conceived as a whole object.

The better half of KP's development goal is to provide the knitters with solutions for this problem - in other word, to upgrade the knitter's notation into an Object Oriented Language.

6. KP Language Snippets

A KP source code defines a knitted item. The symbols it uses are similar to the ones from common knitter's notation. Other symbols are resembles to the ones from C and its cousins, such as C++, java, and sometimes awk.

This is a very simple knit pattern:

```
My_Pattern_1{
	k 20 6
}
```

this makes 20 x 6 stitches of knit (knit is a name of a kind of stitch). This one is a little more complex:

```
My_Pattern_2{
HEAD:
	k 10
	p 10
	k 14
	p 14
	k 18
	p 18
TAIL: }
```

Alternatively, this is:

```
{HEAD:
	kkkkkkkkkk \cl
	pppppppppp \cl
	pppppppppppppp \cl
	kkkkkkkkkkkkkk \cl
	pppppppppppppppppp \cl
	kkkkkkkkkkkkkkkkkk \cl
TAIL:}
```

This makes alterative rows of knit and pearl, incremented by 4 after each round trip. which can be expressed as:

```
{HEAD:
	for(int i = 10; i<=18; i+=4;){
		k i
		p i
	}
TAIL:}
```

Naming of the object is optional, but if named, objects can be used in other objects. This is a vertical Concatenation of the two object:

```
My_Patter_3{
       My_Pattern_1
       My_Pattern_2
}
```

which is equivalent of this:

```
{
       kkkkkkkkkkkkkkkkkkk \cl
       kkkkkkkkkkkkkkkkkkk \cl
       kkkkkkkkkkkkkkkkkkk \cl
       kkkkkkkkkkkkkkkkkkk \cl
       kkkkkkkkkkkkkkkkkkk \cl
       kkkkkkkkkkkkkkkkkkk \cl
       kkkkkkkkkk \cl
       pppppppppp \cl
       pppppppppppppp \cl
       kkkkkkkkkkkkkk \cl
       pppppppppppppppppp \cl
       kkkkkkkkkkkkkkkkk \cl
}
```

(this can be knit, but looks rather ugly...)