Challenges Facing a High-Level Language for Machine Knitting

Lea Albaugh James McCann
Disney Research Pittsburgh
{lea.albaugh, jmccann}@disneyresearch.com

Knitting is the process of creating textile surfaces out of interlocked loops of yarn. With the repeated application of a basic operation – pulling yarn through an existing loop to create a new loop – complicated three-dimensional structures can be created [4]. Knitting machines automate this loop-through-loop process, with some physical limitations arising from their method of storing loops and accessing yarns [1, 3]. Currently, knitting machines are programmed at a very low level. Projects such as AYAB [2] include utilities for designing knit colorwork, but only within a limited stitch architecture; designers working in 3D usually do so via a set of pre-designed templates [4].

In this talk, we will discuss our current progress in understanding what machines can knit, and how this understanding has informed the design of our low-level knitting machine language. We also describe problems that face any high-level knitting description language, and speculate on ways they might be addressed.

1. An (Abstract) Knitting Machine

At its most basic level, a knitting machine must store loops and pass new loops of yarn through its stored loops. Modern v-bed knitting machines store loops on two opposing *beds* (rows) of hook-shaped *needles*. Between the two beds, *yarn feeders* bring yarn into the system.

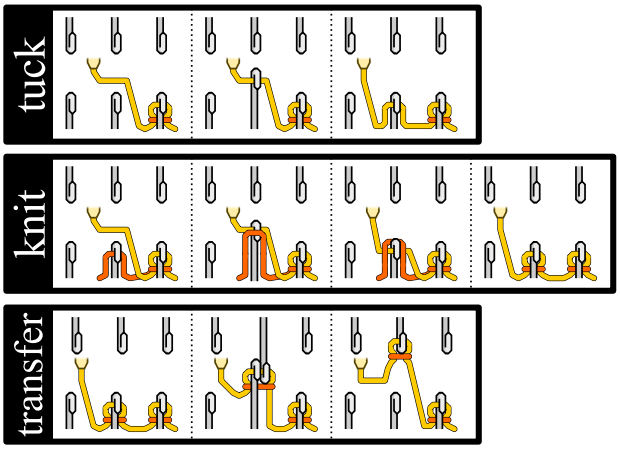

Figure 1. The basic knitting machine operations, viewed from above. Tuck adds a loop to a needle. Knit pulls a loop through (and drops) the current loops on a needle. Transfer hands the loops on a needle to the opposing needle.

Each needle can form and hold loops, as well as transfer its currently held loops to the needle across from it, Figure 1. The alignment of the beds can be changed within machine limitations, allowing loops to be transferred to various offsets.

Once a loop has been dropped, it can no longer be accessed by the machine. A loop can be dropped explicitly, and a loop is always dropped when a new loop is pulled through it unless a special split stitch (which combines a knit with a transfer) is used.

2. What is Machine-Knittable?

The basic operations above can produce a limited number of local structures. In fact – and we state this without proof – the local structure of any machine-knitted object can be described by a directed acyclic graph containing eight types of structure nodes (yarn in, yarn out, drop, miss, tuck, knit, split, stack), and two types of connection (yarn, loop), Figure 2.

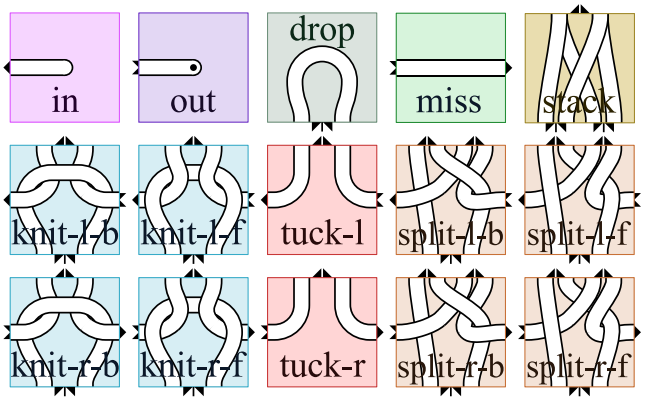

Figure 2. The local structure of a machine-knit object can be described by an acyclic graph containing eight types of nodes, some of which have multiple orientations. Connections on the left and right are yarns, top and bottom are loops. Each connection point on a node must be connected to exactly one other node.

Unfortunately, this is not a complete description of a knit fabric, since it only tracks the behavior of loops, and im-

portant structural connections can be made without loops. Additionally, this description is not unique: portions of the graph can flipped horizontally without changing the described topology.

But now that we have a description of a knit object that isn't a program that produces said object, we can formulate an interesting problem:

2.1 The Knit Scheduling Problem

Given a description of the structures in a knit object, can the object be knit? If so, how?

This boils down to assigning time slices on needles to operations, subject to a few constraints:

1. Yarn isn't particularly stretchy, so subsequent knits, tucks, or misses along the same yarn should happen g needles apart. (Where the gauge, g, is fixed globally to one or two.)
2. Stitches must be of the appropriate orientation; however, there's a choice to be made here, since a right-to-left stitch formed on a front bed needle is a rotated version of a left-to-right stitch formed on a back bed needle.
3. Transfer operations don't appear in the structure, and will need to be inserted as needed. However, moving loops between needles is tedious and error-prone, and machines can only transfer a limited distance. So any stacking operations might require several transfer operations between needles, and all such needles will need to be clear of other loops.

To further complicate matters, the limits of these assignments are machine- and yarn-dependent. Code that works fine with one yarn on one machine might be incompatible with another machine or perhaps even fail with a different yarn on the same machine.

2.2 Tangling

As noted above, yarns (and yarn feeders) can become entangled with each-other depending on the positions where knits are formed. These tangles can serve an important structural function in the finished object, but they can equally be unintended and undesired.

This leads to two further challenges for knit language designers:

1. How can yarn entanglement be detected when error-checking a program?
2. How can users indicate when entanglement is desired?

3. Low-Level Languages For Machines

Current knitting languages – including our own – require knit designers to schedule their own knits. Designers specify actions at particular needles, with a particular yarn, in a particular time order.

Because of this, simple scheduling changes such as deciding to knit on the front bed instead of the back or deciding to knit two tubes side-by-side instead of one after the other can have dramatic impact on the code or even runtime of a program without changing the result at all.

In addition, programs may not be portable between machines and yarns.

4. High-Level Knitting Languages

These low-level approaches represent the machine operations explicitly, and the produced knit structure only implicitly. A high-level knitting language would allow users to act on structural descriptions of a knit, leaving the low-level machine scheduling task to a compiler.

Such a language may be designed so that programmers can specify knits at the macro-scale (in terms of tubes, sheets, and even complete objects) – rather than at a stitch level. However, stitch-level control will still be important for defining surface details, like ribbing or lace.

To allow code written at the stitch level to communicate with code written at the structure level, a high-level knitting language will need to be able to describe not only the possible schedules of various stitches, but also the degrees of freedom in those schedules. In addition, a type theory will need to be developed to describe how and when high-level knitting code can be composed; since many knit items feature basic features (e.g. cuffs) for which modules should be developed.

These problems may seem difficult from a general languages standpoint, but we believe that because knitting programs produce a finite output – at most a few million stitches – and operate in a limited domain – at most a few thousand needles – powerful analysis tools can be brought to bear that would otherwise be impossible to use.

References

[1] L. Albaugh. From text to textiles: !!Con 2015, 2015. URL `http://www.instamatique.com/blog/from-text-to-textiles-con-2015/`.

[2] A. Y. A. Beautiful. Ayab - all yarns are beautiful, 2014. URL `http://ayab-knitting.com/index_en.html#features`. [Online; accessed 4-September-2015].

[3] W. Choi and N. B. Powell. Three dimensional seamless garment knitting on v-bed flat knitting machines. *Journal of Textile and Apparel, Technology and Management*, 4(3):1–33, 2005.

[4] J. Underwood. The design of 3d shape knitted preforms, 2009. PhD Thesis.