Declarative Bioengineering

In any engineering discipline, it is essential that there is a standardization of components and the ability to abstract away complexity. However, for a long time the field of bioengineering has lacked this ability - one reason being that biological systems tend to be very intertwined in complex and unknown ways.

In 2003, the presumption of complexity unique to biology was challenged in the final report of the Synthetic Biology study at MIT [1]. This study greatly influenced the field of synthetic biology over the next 16 years, with many of its predictions coming true.

A particular claim is made in that paper - that “A scalable development path for engineering biology can be realized by combining (i) component standardization, (ii) substrate and component abstraction, and (iii) design and fabrication decoupling.” In particular, we will be focusing on the idea of component abstraction. The paper further claims a “need to promote the characterization and representation of standard biological parts in ways that insulate relevant physical characteristics from overwhelming physical detail.”

In 2008, a concrete example of this idea was published [2]. In it, a “device” was built from standardized DNA parts, which themselves had standardized characterization. This particular example, however, makes a fundamental flaw: it does not abstract function from sequence, it only hides it behind another layer.

In order to realize true component abstraction, functions of biological devices must be defined separately from their implementation. Practically, that means devices are never defined as a sequence. Devices should be defined as objective measurable outputs of genetic systems. For example, BBa_F2620 (the device built in the 2008 paper) would not have a datasheet, the device itself IS the datasheet, with BBa_F2620 being a possible instantiation (or assembly) of that device.

This then creates a problem: how does one go from standard DNA parts to a functional device in a living creature? To accomplish that, I propose the development of “genetic architectures”. For example, the architecture for a GFP expression cassette could be “`A promoter - a RBS - a GFP - a terminator`”, which itself is an extension of the general expression architecture, “`A promoter - a RBS - a CDS - a terminator`”.

The most important part of genetic architectures is that they fit logical patterns, which may be formalized using Logic Programming [3]. This then fits the final part in the puzzle of making genetic engineering into a type of language: While DNA parts have been standardized (words) and measurements have been standardized (sentences/intention), genetic architectures finally standardize grammar. This language can translate human intent into operations done by computers and robots to create biology.

I wish to change the engineering of biology from an imperative programming language (ie, how to accomplish a goal) to a declarative programming language (ie, what goal to accomplish), due to the simple observation that humans are good at “what” and software is good at “how”.

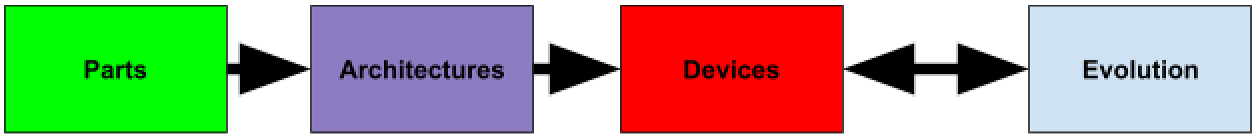

Fig 1. The abstraction layers behind the proposed declarative language. Directed evolution, at the end, can be used to refine devices.

I propose the creation of a simple FreeGenes collection that can directly be used as the test set for creation of a declarative bioengineering language. It will include a few different sugar inducible E.coli promoters, a few different BCDs[4] and RBSs, a few different fluorescent proteins, and a few different terminators. The objective would be to define a simple generic platform for using this language, and then use that platform to implement sugar biosensor devices. This platform could then be used to implement an arbitrary number of architectures and devices from the FreeGenes part repository.

This collection of parts was initiated by Keoni Gandall.

There are no intellectual property concerns surrounding this collection.

PS: There are many aspects that I did not mention here, for example, for outputs of genetic architectures to be sorted using neural nets / statistics trained from previous experiments, or for device instantiations to be refined using directed evolution. However, within the framework I am proposing, there are direct paths to implementing all those aspects within code.

[1]
https://dspace.mit.edu/bitstream/handle/1721.1/38455/SyntheticBiologyStudy.pdf?sequence=1&isAllowed=y
[2] https://sci-hub.tw/https://doi.org/10.1038/nbt1413
[3] https://en.wikipedia.org/wiki/Logic_programming
[4] https://sci-hub.tw/https://doi.org/10.1038/nmeth.2404