Draft - Amanda Clare - 29/9/15

Case for Support: An executable language for change in biological sequences

Description of the Proposed Research

We propose to develop an executable language and an algebra for the representation of change (differences) in biological sequences. This language will allow us to describe a laboratory's strain collection, describe the consequences of decisions in synthetic biology, query a plasmid collection, compare genomic differences of two sequences, and represent these differences at various levels of comprehension. In this work we want to concentrate on the representation of change, using theory from software version control systems. We also plan associated public engagement works about bioinformatics, biological sequences and computing languages.

1 Background

Representing sequences

The representation of biological sequence information (such as DNA) is important to our understanding of synthetic biology and genomics. We need to represent sequence changes and their consequences. We can now modify, replace and delete parts of genomes. We can engineer small DNA components that can be taken up by living organisms. We can construct whole genomes from scratch. We can compare sequences with each other to discover differences, and we can even compare the gigabases of whole genomes across species.

At the moment we have no unifying representation of sequence change or difference. When individual labs need to catalogue the strains, species, plasmids and other components that they hold in stock, a variety of databases and spreadsheets are commonly used. A new researcher joining a lab can look back at the records, and find that, for example, although it made sense at the time for lab members to describe their collection of different yeast strains simply by the name of the single gene that had been deleted, now the lab uses multiple gene deletions and insertions in its experiment, and these have been made in multiple host strains, and the documentation is no longer sufficient.

The languages and formats we currently use for representing DNA sequences vary according to the domain. For genome sequences the FASTA file dominates, a plain text representation of every single base. For sequencing output, FASTQ describes the sequence and also provides quality scores. For short reads, SRA is the NCBI's preferred format, providing compression and indexing, and metadata that describe how the reads were produced. For genome modifications, we have more abstract levels of notation that give the name of the parental strain, and a list of alterations, such as YSBN9 (Mat**a** *ura3*-Δ) to represent a new yeast strain whose parental strain was the 9th produced by the YSBN consortium, is haploid with Mat**a** mating type, and has the ura3 gene deleted. We can indicate disruption of one gene by the insertion of another (ade6::URA4), and replacement of one gene by another (ade6Δ::URA4) [2]. But partial disruption, and complex replacements are difficult to make clear in this language. Other notations exist for single nucleotide polymorphism (SNP) alterations, for recognising variation in short sequence repeats and for storing and operating on sequence alignments.

Representing changes

Computer scientists have also had to face the problem of recording repeated modification to sequences, when storing code. Version control is now a widely accepted standard practice for software engineers. Changes are recorded for posterity, and can be compared, inspected and revoked. Most version control systems (SVN, Git, etc) maintain a history of versions. They allow remote code to be updated to a particular version, and they allow two programmers working on different versions to merge their code and create a single unified version. The Darcs version control system introduced a theory of patches as a new way to record change [17]. Darcs is a change control system rather than a version control system. It uses "patches" to represent the changes between one body of code

and another. Patches can be recorded, and then applied in an order that can be independent of the history of the project, but instead is dependent on the contents of the patch. The theory of patches describes an algebra of operations that can manipulate and apply changes. This patch theory for change control has since begun to be investigated by others [6, 19] and inspires the language and formalism we will need to describe the changes that can be made to biological sequences.

When we represent change as patches, we can discuss the rules of application of patches. If patch A is independent of patch B, then they can be applied in either order, and we can apply just one patch without the other. If patch B is dependent on patch A then this is no longer true. A patch may be able to be repeatedly applied (such as an extension of a microsatellite repeat region), or it may only be applicable once (such as the deletion of a specific gene). A patch may have an inverse operation which can restore the sequence to its previous state. The effect of the application of a series of patches can be calculated and reasoned about.

The description of a patch and its effect is also very important to formalise. A gene deletion may be described at various levels. At the lowest level it can be described by the base changes (a description of the base(s) removed and/or the bases that are inserted). This description may be relative to various elements: relative to the start codon, relative to the coding sequence, relative to the start of the chromosome, or relative to the genome. Our notation must enable an unambiguous low level description, but we must also be able to translate this into the various higher level views that allow us to understand the consequences (such as "deletion of the URA3 gene", "frameshift within the URA3 gene cased by a base insertion, likely disruption of this gene", "replacement of the promoter for URA3 with a TetO-CMV promoter").

We must also be able to scale the language and its operators so that we can represent all sizes of sequence from tens of bases up to billions of bases: primers, genes, details of plasmid construction, chromosomes and whole genomes (the wheat genome has 17 billion base pairs). We wish to be able to use the same language to represent and compare the differences between two primers, and also to ask about the different SNPs in two genomes. Storing a compressed version of the underlying common sequence and then separately storing the patches that describe the changes enables fast comparison of patches without needing to process the whole genome.

A Domain Specific Language (DSL), is a programming language (or subset of a programming language) that allows domain-specific terminology to be used directly and easily. The aim is to create a language that is clear and concise and can be used by non-programmers. Examples of DSLs include languages for the description of financial contracts by investment banks [16], for the description of quantum physics [10], for SAT-solving and circuit design[18], and for XML manipulation [20]. An embedded DSL (EDSL) is a domain-specific language that is embedded within a more expressive programming language. Haskell is a an expressive programming language that provides good support for the construction and manipulation of EDSLs. It provides strong type checking that can be used to check that expressions in the DSL are valid, higher order functions (and precedence definitions for operators) that allow natural compositions of terms, libraries for parsing, and excellent support for the definition of new datatypes. A practical and usable embedded DSL in Haskell, would allow the clear description, analysis and comparison of change in biological sequences.

2 Academic Impact

The language, its associated tools, and the example applications will be available for synthetic biology researchers and bioinformaticians to use. We aim to provide this community with a means to clearly represent and communicate sequence changes. The publication of software and data along with research papers is now strongly encouraged, and this work helps those who need to describe their sequence changes, publish their data in a clear and formalised language.

For ordinary laboratories that are currently cataloging their strain collections via a succession of postdocs and students with office software such as Excel and Access, we will be providing a useful tool, allowing them to query their collection and bringing their descriptions up to date.

In computer science we have a real and complex application to test the theories of DSL design,

change control theory and even programming language theory. This application is likely to benefit from stronger typing than that used in conventional languages. It will raise questions such as: Would the use of dependent types give us greater power for validation and derivation of consequences [1]? Will the existing solutions to the problems when sharing structure and using recursion in DSLs be applicable to this domain and scalable to the description of large changes in large sequences [14, 12]? We will take advantage of current research interest in these areas, providing a practical demonstration of the theories and proposals in DSL research.

3 Research Hypothesis and Objectives

The aim of this research is to produce an executable language for the representation of changes to biological sequences. The research hypothesis is that this language can be mathematically well defined and at the same time can be practical and applicable. We aim to use the language to unambiguously and clearly describe sequence modifications, reason about their effects, compare multiple sequences to highlight their differences, and efficiently store this information.

The objectives are:

1. A design for the components, notation and operators for such a language, and the algebraic theory of the operators.
2. A compiler and execution system for such a language, including optimisation such as sequence compression, and integration with version control.
3. A human-readable representation of the sequences, changes, and effects of the changes.
4. Three demonstrator applications that process and store sequence information, and clearly show the usefulness.

The demonstrator applications will be:

- A bacterial strain storage utility, aimed at small to medium sized labs and allowing them to keep track of which bacterial strains they have, their associated metadata (where, when, who, what), and to compare and inspect strains within their collection.
- A user-friendly language to query a plasmid repository for plasmids with specific components and functionality
- Tools for conversion to and from many other standard sequence file formats.

4 Programme and Methodology

The workpackages that make up the programme of work are as follows:

WP1: Language Design

This will meet objective 1 and will involve the design of:

- terms, operators and rules for combination of expressions
- the information required to describe a change (or patch)
- rules for the calculation of the effects of a change to a sequence
- various levels of abstraction for translation/presentation

The deliverable from this workpackage will be a publication describing the language design and the issues that we can can foresee before implementation.

WP2: Implementation

This will meet objectives 2 and 3. The language will be implemented as an embedded DSL, and we plan to use Haskell (or possibly a derivative such as Agda) for this purpose. This will involve:

- I/O (parsing and pretty-printing)
- compression (integration of existing compression methods and software with this framework, or development of more suitable compression where required)

- compilation and optimisation (taking note of common problems in the implementation of DSLs, such as sharing and recursion)
- storage of changes/patches
- application of a series of changes/patches
- consequence generation (calculating the effects of a change to a sequence)
- basic debugging and linting/validation tools (providing the user with tools that give an understanding of the reasons for the calculation of the effects to a sequence, and checking that the sequence description is well formed).
- creating human-readable output formats

The outputs will be working software tools (freely available under open source licences). Here we will also produce a publication for the functional programming community describing our implementation experiences and insights.

WP3: Demonstrator applications

This will meet objective 4. The three demonstrator applications enable the language to be used immediately.

- File conversion utilities will be vital to promote the adoption of this work.
- A bacterial strain storage utility will be relevant and practical for many small labs globally, who often manage their collections through spreadsheets, Word documents or old bespoke software that they once had a budget to maintain.
- A plasmid repository search language will demonstrate the benefits of describing and accessing small components of sequences that are useful in their own right, and the benefits of sequence comparison to understand the effects of two plasmids with different components.

The outputs will be the demonstrator applications (freely available under open source licences). We will publish descriptions of these in bioinformatics journals.

WP4: Public engagement

We have previously produced the web-based Genome Game[4] which we have used extensively for public engagement discussions of bioinformatics and genomics, particularly with children. This is one of the few bioinformatics games in existance, and is available bilingually (English/Welsh). We wish to continue to demonstrate any new work in a way that also opens dialogue with the public.

DNA sequence has always been the poster-child of molecular biology when it comes to public outreach, so we should be able to use this work to make exciting demonstrations to take into schools and to entice the media to take an interest. The ethical considerations for sequence editing raise multiple questions that the public need to be aware of. We would like to produce audio and visually animated representations of sequence differences for the general public. This could be done using Haskell libraries (e.g. Haskore, Gloss, Diagrams), or languages like Elm, or simply Javascript/HTML5 for web demos. We would provide sequences and sequence differences for such representations, and then ask members of the public to compare sequences by music and by animation. In particular, our Computer Science Department Robot Orchestra would be extremely eye catching when playing some of these sequences. We can also use these output formats to illustrate topical and serious news stories about genomic differences, such as comparing the invading ash dieback fungus with the similar native species, the BRCA1 and 2 gene differences that caused Angelina Jolie to have pre-emptive surgery, or to demonstrate how DNA barcodes can be used to distinguish between the various plant species found in Wales[7].

Methodology

Algebra and DSL: Domain specific languages embedded in a host language give us a formal notation that looks human-readable, using terminology familar to biologists and bioinformaticians, but is at the same time executable (as a computer program). Being executable means that we can calculate results, manipulate the terms, and apply different functions (such as comparison, reformatting for

output, or testing properties), all within the host programming language. Being a formal language means that we can specify the rules for the structure of valid terms and define the operations that can be performed in an unambiguous manner.

Change control: We will take the theory of patches as our starting point, and make an analysis of how the rules of patch application for software source code change compare to the rules of patch application for biological sequence change. We will investigate and catalogue the known dependencies and potential consequences of alterations to sequence.

Compression: Researchers have already produced genome compression algorithms that specialise in compression against a reference sequence, such as [9]. Their intention is usually to save space rather than to describe change. New algorithms mainly concentrate on ways to detect lengths of DNA without differences, to be efficient at describing the parts that differ, or to decide what information can be discarded [5]. Our intention is not to save space, but instead to characterise the differences in order to make them applicable and descriptive entities. However, we will make use of existing compression algorithms in order to be reasonably efficient. In order that we can still compute and present the effects of the changes, we will need to use a compression format that allows queries for the region of interest, such as self-indexing compression [13].

Related work: Perhaps the most useful starting point for the description of sequence in our work will be the ontology of biological sequences [11] which provides sequence components and their relations and axioms, and is based on the Sequence Ontology [8]. Another useful and practical source is the Biobricks Foundation, which hosts a Registry of Standard Biological Parts. In this registry, users can search for parts such as promotors, terminators, plasmid backbones and protein coding sequences. The repository is set up so that biologists can request the part as an aliquot of DNA in a plate. They can then use the knowledge of the sequences of these listed parts to engineer (synthesise) larger components which can be inserted into organisms to create novel biological material. We would like to be able to represent these parts, and the subsequent constructions that can be synthesised using them. Similarly, GenoCAD is a software tool providing a library of DNA parts that the user can select and use in the design of a synthetic molecule. They offer no comparison or change control features, but they do offer simulation of the final function of the molecule by translating the DNA using attribute grammars into a network model suitable for execution on COPASI [3]. PyDNA [15] is an interesting new language implemented in Python, which allows the user to simulate a process of sequence synthesis or Gibson assembly, checking overlap areas, primer functionality and calculating regions that will join or use homologous recombination.

Project management: We will have weekly project meetings, involving the PI, co-I and postdoc to discuss the issues as they arise. The postdoc will have the support of the Bioinformatics and Computational Biology Research group within the Dept of Computer Science, and also have easy access to the biologists in the neighbouring Institute of Biological, Environmental and Rural Sciences, and our shared bioinformatics seminars and workshops. Aberystwyth University offers ongoing staff training modules throughout the year, which we will all attend, to help with presentation skills, time management, and other project management activities. The Research Office in Aberystwyth provides dedicated support for finance, project management and other research administration. The workplan for the project is given in the attached page.

5 National Importance

TODO

References

[1] T. Altenkirch, C. McBride, and J. McKinna. Why dependent types matter. Technical Report `http://www.cs.nott.ac.uk/~txa/publ/ydtm.pdf`, 2005.

[2] D. Burke, D. Dawson, and T. Sterns. *Methods in Yeast Genetics: A Cold Spring Harbor Laboratory Course Manual*. Cold Spring Harbor Laboratory Press, 2000.

[3] Y. Cai, M. W. Lux, L. Adam, and J. Peccoud. Modeling structure-function relationships in synthetic DNA sequences using attribute grammars. *PLoS Comput Biol*, 5(10):e1000529, 2009.

[4] A. Clare. The Genome Game. http://genome-game.dcs.aber.ac.uk/, 2013.
[5] G. Cochrane et al. The future of DNA sequence archiving. *GigaScience*, 1:2, 2012.
[6] J. Dagit. Type-correct changes - a safe approach to version control implementation. Master's thesis, Oregon State University, 2009.
[7] N. de Vere et al. DNA barcoding the native flowering plants and conifers of Wales. *PLOS One*, doi: 10.1371/journal.pone.0037945, 2012.
[8] K. Eilbeck et al. The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biology*, 6(5):R44, 2005.
[9] M. H-Y. Fritz et al. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research*, 21:734–740, 2011.
[10] A. S. Green et al. An introduction to quantum programming in Quipper. In *Reversible Computation 2013*, volume 7948, pages 110–124. Springer, LNCS, 2013.
[11] R. Hoehndorf et al. The ontology of biological sequences. *BMC Bioinformatics*, 10:377, 2009.
[12] O. Kiselyov. Implementing explicit and finding implicit sharing in embedded DSLs. In *Proceedings of DSL 2011, IFIP Working Conference on Domain-Specific Languages*, 2011.
[13] V. Mäkinen et al. Storage and retrieval of individual genomes. In *RECOMB 2009*, 2009.
[14] B. C. d. S. Oliveira and A. Löh. Abstract syntax graphs for domain specific languages. In *Proceedings of the ACM SIGPLAN 2013 workshop PEPM '13*, pages 87–96, 2013.
[15] F. Pereira et al. Pydna: a simulation and documentation tool for DNA assembly strategies using python. *BMC Bioinformatics*, 16:142, 2015.
[16] S. Peyton Jones, J-M. Eber, and J. Seward. Composing contracts: an adventure in financial engineering. In *ICFP 2000*, 2000.
[17] D. Roundy. Darcs: Distributed version management in Haskell. In *SIGPLAN Workshop on Haskell*, 2005.
[18] D. Stewart. Combining languages and SMT solvers: an EDSL study. IFIP Working Group 2.8, 2011.
[19] W. Swierstra and A. Löh. The semantics of version control. In *Proceedings of Onward 2014*, 2014.
[20] M. Wallace and C. Runciman. Haskell and XML: Generic combinators or type-based translation? In *ICFP 1999*, 1999.