Journal of Applied Mathematics & Bioinformatics, vol.2, no.2, 2012, 65-79
ISSN: 1792-6602 (print), 1792-6939 (online)
Scientific Press, 2012

Nucleic Acids Data Sequencing using Higher Order Logic-A Suggestion of Basic Computational Framework Towards Bio-Sensors and Gene-Chips Design, Implementation and Verification*

D.N.T. Kumar[1], Yossi Kost[2], Hui Qiao[3] and Qufu Wei[4]

Abstract

In this current research paper we highlight the usage of Higher Order Logic (HOL) to compute the sequences of Nucleic acids, DNA and RNA and illustrate the usefulness of higher order logic in computing the sequences for bio-sensor, probes, tags and gene-chip applications. What ever the application in question,sequences always form the heart and soul of bio-information processing in such demanding and promising tools. Higher order Logic provides a comfortable way of solving or computing the sequencing problems and related issues in the advancement of bio-informatics,hence we have focused in this interdisciplinary paper on HOL and its applications, in the design of such bio-

[1] Jiangnan University, China and Ben-Gurion University, Israel, e-mail:tejdnk@gmail.com
[2] Ben-Gurion University, Israel, e-mail:kost@bgu.ac.il
[3] Jiangnan University, China, e-mail:qfwei@jiangnan.edu.cn
[4] Jiangnan University, China, e-mail:qfwei@jiangnan.edu.cn

Article Info: *Received* : August 11, 2012. *Revised* : September 14, 2012
Published online : September 15, 2012

devices. Further more we are of the opinion that HOL is fast becoming the de-facto verification and computational software tool, in verifying the electronic hardware in demanding applications, the same methodology could be extended to bio-inspired systems, bio-systems, theoretical biology etc. to explore the untouched relevant areas of computational design paradigms. This is one of the pioneering research efforts in this highly promising domain, to adopt and adapt HOL as the design tool to model and develop sequencing, in Bio-Computing or Bio-Sensing platforms.

Mathematics Subject Classification : 03B15, 92-04, 92-08
Keywords: HOL, DNA, RNA, Gene-Chips, Design, Theoretical Biology, Bio-Sensors, Probes, Tags, Bio-informatics, Computation, Design

1 Introduction

Introduction and Motivation on Bio-Computing using Formalisms & HOL

We are constantly searching for biological models, that are hardly sufficient by querying raw model code of a certain interchange format. Further exploration deserves merit, it is worthwhile to include additional information about a bio-model:

(1) What is known about a bio-system/model's constituents?
(2) What kind of relations exist between its bio-system constituents?
(3) How does a model behave under certain conditions like sequencing or sensing?

Most of these research questions can be answered by formalisms and applied mathematics. Thus, a thorough annotation of bio-model enhances the semantic description of the modeled system by far and could contribute to sophisticated bio-applications techniques. Efforts for enhanced model annotation, such as the "HOL based models are already applied to some modeling formalisms, e.g. the "Systems

Biology Markup Language" (SBML).The main question for this research work is, how the information available about a bio-model can be gathered, processed and analyzed, and stored in a manner to efficiently retrieve bio-models for design, implement, verify or compute a specified bio-action such as sensing or sequencing. Hence we state here that we have adopted an interdisciplinary approach in computing, the sequences of DNA/RNA sequences by using Mathematics, Computer Science and Biology. This we believe will be the future of DNA based Sensors and Gene-Chips in the near future, [1-8].

2 Basics on HOL/Nucleic Acids/Sequences

2.1 HOL

"We develop the idea of higher-order logic programming by utilizing a higher order logic as the basis for computing. There are, of course, many choices for the higher-order logic that might be used in such a study. If the desire is only to emulate the higher-order features found in functional programming languages, it is possible to adopt a "minimalist" approach, i.e.,to consider extending the logic of first-order Horn clauses — the logical basis of Prolog — in as small a way as possible to realize the additional functionality. The approach that we adopt here, however, is to enunciate a notion of higher-order logic programming by describing an analogue of Horn clauses within a rich higher-order logic, namely, Church's Simple Theory of Types", [1-5, 15].

"Philosophers of mathematics usually divide logic into first-order logic and second-order logic. The latter is a formal basis for all of mathematics and, as a consequence of G¨odel's first incompleteness theorem, cannot be recursively axiomatized. Thus, higher-order logic in this sense is basically a model theoretic study.", [1-11, 14, 15].

"To a proof theorist, all logics correspond to formal systems that are

recursively presented and a higher-order logic is no different. The main distinction between a higher-order and a first order logic is the presence in the former of predicate variables and comprehension, i.e., the ability to form abstractions over formula expressions. Cut-elimination proofs for higher-order logics differ qualitatively from those for first-order logic in that they need techniques such as Girard's "candidats de reductibilite," whereas proofs in first-order logics can generally be done by induction. Semantic arguments can be employed in this setting, but general models (including non-standard models) in the sense of implementation must be considered. To many working in automated deduction, higher-order logic refers to any computational logic that contains typed, terms and/or variables of some higher-order type, although not necessarily of predicate type. Occasionally, such a logic may incorporate the rules of conversion, and then unification of expressions would have to be carried out relative to these rules.", [15].

Information on HOL Software Used

"Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. Isabelle is developed at University of Cambridge (Larry Paulson), Technische Universität München (Tobias Nipkow) and Université Paris-Sud (Makarius Wenzel). See the Isabelle overview for a brief introduction", [1].

2.2 Nucleic Acids DNA/RNA/Protein data sequencing in DNA

There are actually 2 main types of nucleic substances within cell nuclei that process information. DNA is the basic form within chromosomes, that is hard-coded into every cell. RNA is a more temporary form that is used to process subsequences of DNA messages. RNA is an intermediate form used to execute the portions of DNA that a cell is using. For example, in the synthesis of proteins,

DNA is copied to RNA, which is then used to create proteins :

DNA->RNA->Proteins. [1-13, 16]

DNA code is a sequence of chemicals that form information that control how humans are made and how they work. It is a digital code but it is not binary, but quaternary with 4 distinct items. The encoding information in an ordered sequence of 4 different symbols called "bases", typically denoted A, C, G, and T.

Step 1: A: adenosine

Step 2: C: cytosine

Step 3: G: guanine

Step 4: T: thymine

'These 4 substances are the fundamental "bits" of information in the genetic code, and are called "base pairs" because there is actually 2 substances per "bit", as discussed later. Everything else is built on top of this basis of 4 DNA digits.The entirety of human DNA code, called the "human genome", is about 3 million bases in total. Every human being has 2 copies of this code, one copy from each parent, so a human's cell DNA contains a total of around 6 billion bases. In computer terms, this is around 6 Gigabytes of symbols, or more like 1 Gigabyte if compacted, since it's about 2 binary bits of information per A/C/G/T base pair. DNA molecules are linear in a twisted double-helix, with a start and an end, and do not contain any cycles", [12-15, 16].

The structure of DNA and RNA are very similar. They are both ordered sequences of 4 types of substances: ACGT for DNA, and ACGU for RNA. Thus RNA uses the same three ACG substances, but uses U (uracil) instead of T (thymine). The molecules uracil and thymine are only slightly different chemically. In DNA, there is pairing between AT and CG, and in RNA, the pairings are AU and CG, but since RNA is not double-stranded, this pairing is much rarer. Hence, RNA has the 4 substances :

Step 1: A: adenosine

Step 2: C: cytosine

Step 3: G: guanine

Step 4: U: uracil

“Typically, DNA is created from RNA, and this is done by faithfully copying the sequence of base pairs, with the only change converting T to U. Hence, an RNA copy of a DNA sequence encodes the identical information, though it uses a slightly different set of 4 substances. The differences between DNA and RNA are also many. The underlying sugar molecule that traps the 4 bases is different: deoxyribose in DNA, ribose in RNA. DNA is two strands wrapped in a double-helix, but RNA is a single strand”, [1-12, 16].

2.3 Genes: Protein Data Sequences in the DNA Code

Some parts of DNA sequences are known to be purely data. These are the "genes". The best understood aspect of DNA coding is the encoding of amino acid information in genes that is used by the body to synthesize proteins. These are data blocks that represent protein structures.

“All proteins are substances made up of only 20 basic building blocks called amino acids. Proteins are ordered sequences of these 20 amino acids. Another terminology is that an amino acid is a "peptide" and a protein is a sequence of many peptides called a "polypeptide", [8-16].

So how does DNA encode the structure of a protein? It uses triplets of base pairs. There are 4x4x4=64 possible combinations in a base pair triplet, and only 20 amino acids. Some extra codes are used as start and stop signal markers at each end of the data sequence. Other triplets are mapped so that more than one triplet can represent a particular amino acid”, [8-16]. However, the representation is unique across all DNA mapping base pair triplets to the 20 amino acids:

4.1.1. Phenylalanine (Phe): UUU, UUC

4.2.2. Leucine (Leu): UUA, UUG

4.3.3. Isoleucine (Ile): AUU, AUC, AUA

4.4.4. Methionine (Met): AUG

4.5.5. Valine (Val): GUU, GUC, GUA, GUG

4.6.6. Serine (Ser): UCU, UCC, UCA, UCG, AGU, ACG

4.7.7. Proline (Pro): CCU, CCC, CCA, CCG

4.8.8. Threonine (Thr): ACU, ACC, ACA, ACG

4.9.9. Alanine (Ala): GCU, GCC, GCA, GCG

4.10.10. Tyrosine (Tyr): UAU, UAC

4.11.11. Histidine (His): CAU, CAC

4.12.12. Glutamine (Gln): CAA, CAG

4.13.13. Asparagine (Asn): AAU, AAC

4.14.14. Lysine (Lys): AAA, AAG

4.15.15. Aspartic acid (Asp): GAU, GAC

4.16.16. Glutamic acid (Glu): GAA, GAG

4.17.17. Cysteine (Cys): UGU, UGC

4.18.18. Tryptophan (Trp): UGG

4.19.19. Arginine (Arg): CGU, CGC, CGA, CGG, AGA, AGG

4.20.20. Glycine (Gly): GGU, GGC, GGA, GGG

In addition, the following triplet codes are considered special:

[i] STOP: UAA, UAG, UGA

[ii] START: AUG (same code as the Methionine amino acid)

2.4 Sequential Calculi

DNA sequencing using HOL is based,on the following definition of sequential calculus.“A sequential calculus gives a logic of sequences (of a well-known formula). The formulae in the sequence are separated by the symbol "||-" which means that the formulae on the right of it, taken conjunctively make true the formulas to the left of it, taken disjunctively. Formulae on the right and left of "||-" are separated by commas”, [1-4, 17].

3 Application of HOL to DNA/RNA Sequencing with basic code

```
(*  Title:      Sequencing/dnasequence.thy
    Author:     Nirmal;Jiangnan University,Wuxi,214122,China/Ben-Gurion University of the Negev,BeerSheva,84105,Israel
    This HOL Code is easily adaptable to RNA sequencing or other bio-applications where sequencing is used.
    e.g- RNA Data Sequences in DNA/Genes: Protein Data Sequences in the DNA Code.*)

header {* Parsing and Printing of dna-sequences/A Novel Approach in Bio-informatics*}

theory dnasequences

imports Pure

uses ("prover.ML")

begin

setup Pure_Thy.old_appl_syntax_setup

declare [[unify_trace_bound = 20, unify_search_bound = 40]]

typedecl o

(* dna -sequences *)

typedecl

 dnaseq'

consts

 dnaSeqO'        :: "[o,dnaseq']=>dnaseq'"

 dnaSeq1'        :: "o=>dnaseq'"

(* concrete syntax *)

nonterminal dnaseq and dnaseqobj and dnaseqcont
syntax

 "_SeqEmp"        :: dnaseq                          ("")

 "_SeqApp"        :: "[dnaseqobj,dnaseqcont] => dnaseq"          ("__")

 "_SeqContEmp"    :: dnaseqcont                      ("")
```

```
 "_SeqContApp"     :: "[dnaseqobj,dnaseqcont] => dnaseqcont"       (",/ __")

 "_SeqO"           :: "o => dnaseqobj"                    ("_")

 "_SeqId"          :: "'a => dnaseqobj"                   ("$_")

type_synonym single_seqe = "[dnaseq,dnaseqobj] => prop"

type_synonym single_seqi = "[dnaseq'=>dnaseq',seq'=>seq'] => prop"

type_synonym two_seqi    = "[dnaseq'=>dnaseq', seq'=>seq'] => prop"

type_synonym two_seqe    = "[dnaseq,dnaseq] => prop"

type_synonym three_seqi  = "[dnaseq'=>dnaseq', dnaseq'=>dnaseq', dnaseq'=>dnaseq'] =>
prop"

type_synonym three_seqe  = "[dnaseq, dnaseq, dnaseq] => prop"

type_synonym four_seqi     = "[dnaseq'=>dnaseq', dnaseq'=>dnaseq', dnaseq'=>dnaseq',
dnaseq'=>dnaseq'] => prop"

type_synonym four_seqe   = "[dnaseq, dnaseq, dnaseq, dnaseq] => prop"

type_synonym sequence_name = "dnaseq'=>dnaseq'"

syntax

  (*Constant to allow definitions of SEQUENCES of formulas*)

  "_Side"        :: "dnaseq=>(dnaseq'=>dnaseq')" ("<<(_)>>")

ML {*

(* parse translation for sequences *)

fun abs_seq' t = Abs ("s", Type (@{type_name dnaseq'}, []), t);

fun seqobj_tr (Const (@{syntax_const "_SeqO"}, _) $ f) =

      Const (@{const_syntax SeqO'}, dummyT) $ f

  | seqobj_tr (_ $ i) = i;

fun seqcont_tr (Const (@{syntax_const "_SeqContEmp"}, _)) = Bound 0

  | seqcont_tr (Const (@{syntax_const "_SeqContApp"}, _) $ so $ sc) =

      seqobj_tr so $ seqcont_tr sc;
```

```
fun seq_tr (Const (@{syntax_const "_SeqEmp"}, _)) = abs_seq' (Bound 0)
  | seq_tr (Const (@{syntax_const "_SeqApp"}, _) $ so $ sc) =
      abs_seq'(seqobj_tr so $ seqcont_tr sc);
fun singlobj_tr (Const (@{syntax_const "_SeqO"},_) $ f) =
  abs_seq' ((Const (@{const_syntax SeqO'}, dummyT) $ f) $ Bound 0);
(* print translation for dna-sequences *)
fun seqcont_tr' (Bound 0) =
      Const (@{syntax_const "_SeqContEmp"}, dummyT)
  | seqcont_tr' (Const (@{const_syntax SeqO'}, _) $ f $ s) =
      Const (@{syntax_const "_SeqContApp"}, dummyT) $
        (Const (@{syntax_const "_SeqO"}, dummyT) $ f) $ seqcont_tr' s
  | seqcont_tr' (i $ s) =
      Const (@{syntax_const "_SeqContApp"}, dummyT) $
        (Const (@{syntax_const "_SeqId"}, dummyT) $ i) $ seqcont_tr' s;
fun seq_tr' s =
  let
    fun seq_itr' (Bound 0) = Const (@{syntax_const "_SeqEmp"}, dummyT)
      | seq_itr' (Const (@{const_syntax SeqO'}, _) $ f $ s) =
          Const (@{syntax_const "_SeqApp"}, dummyT) $
            (Const (@{syntax_const "_SeqO"}, dummyT) $ f) $ seqcont_tr' s
      | seq_itr' (i $ s) =
Const (@{syntax_const "_SeqApp"}, dummyT) $
(Const (@{syntax_const "_SeqId"}, dummyT) $ i) $ seqcont_tr' s
in case s of
      Abs (_, _, t) => seq_itr' t
    | _ => s $ Bound 0
```

```
  end;

fun single_tr c [A, G] =

  Const (c, dummyT) $ seq_tr A $ singlobj_tr G;

fun two_seq_tr c [A, G] =

Const (c, dummyT) $ seq_tr A $ seq_tr G;

fun three_seq_tr c [A, G, C] =

Const (c, dummyT) $ seq_tr A $ seq_tr G $ seq_tr C;

fun four_seq_tr c [A, G, C, T] =

  Const (c, dummyT) $ seq_tr A $ seq_tr G $ seq_tr C $ seq_tr T;
fun singlobj_tr' (Const (@{const_syntax SeqO'},_) $ fm) = fm

  | singlobj_tr' id = Const (@{syntax_const "_SeqId"}, dummyT) $ id;

fun single_tr' c [A, G] =

Const (c, dummyT) $ seq_tr' A $ seq_tr' G;

fun two_seq_tr' c [A, G] =

Const (c, dummyT) $ seq_tr' A $ seq_tr' G;

fun three_seq_tr' c [A, G, C] =

Const (c, dummyT) $ seq_tr' A $ seq_tr' G $ seq_tr' C;

fun four_seq_tr' c [A, G, C, T] =

 Const (c, dummyT) $ seq_tr' A $ seq_tr' G $ seq_tr' C $ seq_tr' T;

(** for the <<...>> notation **)
fun side_tr [A] = seq_tr A;
*}

parse_translation {* [(@{syntax_const "_Side"}, side_tr)] *}

use "prover.ML"

end
```

4 Discussion and Analysis

Development of “theory of logic programming”, within higher-order logic has another fortunate outcome. The clarity of presentations of the central results regarding higher-order horn clauses require the use of the “sequent calculus”-resolution and model theory based methods that are traditionally used in the analysis of first-order logic programming are either not useful or not available in the higher-order context. It turns out that the “sequent calculus”, is a quite useful tool for characterizing the intrinsic role of logical connectives within logic programming and a study such as the one undertaken here illuminates this fact. This observation was mentioned, in a more complete fashion in one of the chapters by Loveland and Nadathur in [8-15].

“*Given the increasing demand for practical and low-cost analytical techniques, bio-sensors and Gene-Chips have attracted attention for use in the quality analysis of drugs, medicines, and other analytes of interest in the pharmaceutical area. Bio-sensors and Gene-Chips allow quantification,not only of the active component in pharmaceutical formulations, but also the analysis of degradation products and metabolites in biological fluids*”, [11-15].

***Please note: We are not going to discuss the underlying concepts in detail because the current paper is intended as a Technical Note. Hence only the basic framework is discussed.

5 Conclusion

Finally we conclude that the DNA is digital, but is quaternary, not binary. DNA is a base-4 code using the digits A, C, G and T. Proteins are a base-20 code using the 20 amino acids. DNA represents a protein has an ordered sequence of base-4 triplets, using 64 possible values to 20 amino acids. Some DNA sequences are ignored: introns. Each Chromosome has sub-sequences of DNA bases that

encode particular features, and these are called "genes". Thus genes are not independent molecules, but are abstract sequences within chromosomes. All genes have different lengths. Genes are too small to be physically seen on a microscope, but are analyzed using indirect chemical, molecular, and computational methods. This paper describes two experiments in program design, coding and verification, carried out within two automated proof assistants, namely the HOL (Higher Order Logic) system and Isabelle.

Basic approaches to programming language semantics are described. Theories and tactics for proving the correctness of programs written in small functional and imperative language ML, are then constructed within HOL and Isabelle Programming environment.

Acknowledgements. We, sincerely thank all the members, involved in making this paper a possibility. Further, we thank Jiangnan University, China and Ben-Gurion University of the Negev, Israel, for their research support and for providing conducive research environment. We finally thank the authors, of some of the published papers, who willingly gave us permissions to reproduce some materials, from their research papers and also encouraged us to write this paper. We declare further that, we have no competing financial interests in the method presented in this paper. The authors strictly abide by Open Source Software policies and agreements.

The following laboratories in China and Israel participated in the R & D Program.

Key Laboratory of Eco-Textiles,Graduate School of Textiles&Clothing
Ministry of Education, Jiangnan University,Wuxi, 214122, P.R.China.
SCME Jiangnan University,Wuxi 214122, Jiangsu Province, P.R.China.
Laboratory of Controlled Release Systems & Gene Therapy,
Dept of Chemical Engineering,BGU Israel.

National Institute for Bio-Technology of the Negev, BGU, Israel.
The Ilse Katz, Center for Meso and Nanoscale Science and Technology,
Ben-Gurion University of the Negev, BeerSheva, Negev, 84105, Israel.

References

[1] http://www.cl.cam.ac.uk/research/hvg/isabelle/, 2012

[2] http://www.cl.cam.ac.uk/~acjf3/, 2012.

[3] http://cs.anu.edu.au/student/comp8033/hol.html, 2012

[4] http://www.ccs.neu.edu/home/viola/classes/gems-08/lectures/le11.pdf, 2012.

[5] A. Sorribas, J.M. Muiño and E.O. Voit, GS-distributions: a new family of distributions for continous unimodal variables, *Computational Statistics and Data Analysis*, **50**, (2006), 2769-2798.

[6] A. Sorribas, B. Hernandez-Bermejo, E. Vilaprinyo and R. Alves, Cooperativity and saturation in biochemical networks: A saturable formalism using Taylor series approximations, *Biotechnol Bioeng*, **97**(5), (2007), 1259-1277.

[7] R. Alves, E. Vilaprinyo, B. Hernandez-Bermejo and A. Sorribas, Mathematical formalisms based on approximated kinetic representations for modeling genetic and metabolic pathways, *Biotechnology and Genetic Engineering Reviews*, **25**, (2008), 1-40.

[8] R. Alves, E. Vilaprinyo and A. Sorribas, Integrating Bioinformatics and Computational Biology: Perspectives and Possibilities for in silico network reconstruction in Molecular Systems Biology, *Current Bioinformatics*, **3**, (2008), 98-129.

[9] R. Alves, E. Vilaprinyo and A. Sorribas, Semi-Automated Reconstruction of Biological Circuits, *Philippine Information Technology Journal*, **1**, (2008), 32.

[10] A. Sorribas, E. Vilaprinyo and R. Alves, Approximate formalisms: does

anything work?, *Philippine Information Technology Journal*, **1**, (2008), 34.

[11] G. Guillén-Gosálbez, C. Pozo, L. Jiménez and A. Sorribas, An Outer Approximation Algorithm for the Global Optimization of Regulated Metabolic Systems, *Computer Aided Chemical Engineering*, **27**, (2009), 1707-1712.

[12] A. Sorribas, C. Pozo, E. Vilaprinyo, G. Guillén-Gosálbez, L. Jimenez and R. Alves, Optimization and evolution in metabolic pathways: global optimization techniques in Generalized Mass Action models, *J. Biotechnol*, 149, (2010), 141-153.

[13] C. Pozo, G. Guillén-Gosálbez, A. Sorribas and L. Jimenez, (2010), Outer approximation-based algorithm for biotechnology studies in systems biology, *Computers & Chemical Engineering*, **34**, (2010), 1719-1730.

[14] http://bioinformatics.bio.uu.nl/BINF/binf2.pdf.

[15] http://www.lix.polytechnique.fr/~dale/papers/Handbook_Logic_AI_LP.pdf

[16] Joseph Wang, Survey and Summary, From DNA Biosensors to Gene Chips, *Nucleic Acids Research*, **28**(16), (2000), 3011-3016.

[17] http://www.mbph.de/
{ Website of Professor Prof. Dr. Manuel Bremer, Institute of Philosophy, University of Düsseldorf, D-40225 Düsseldorf, Germany}, 2012.