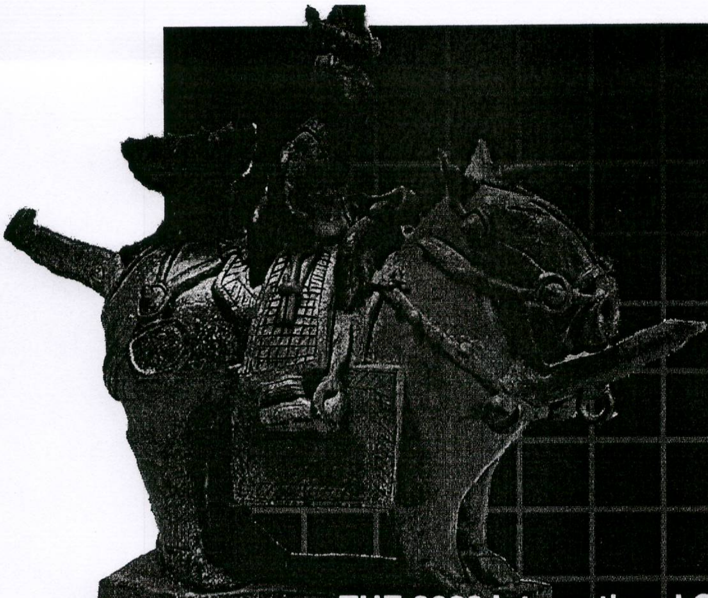

THE 2008 International Conference on
Convergence and Hybrid Information Technology

ICCIT 2008

11 – 13 November 2008
Novotel Ambasador Busan, Busan, Korea

Corresponding Editors:

Dr. Paul P. Wang, Dr. Sungwon Sohn, Dr. Gurpreet Dhillon
Dr. Jungwoo Lee, Dr. Franz I.S. Ko, Dr. Ngoc Thanh Nguyen
Dr. Jun Bi, Dr. Kouichi Sakurai, Dr. Majid Ahmadi
Dr. Yun Ji Na, Dr. Wan-Young Chung

Volume I

IEEE IEEE computer society ETRI AICIT Dynamic BUSAN

IEEE Computer Society Order Number E3407
BMS Part Number CFP0811F-PRT
ISBN 978-0-7695-3407-7
Library of Congress Number 2008928439

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 800 272 6657
Fax: + 1 714 821 4641
http://computer.org/cspress
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: <reprints@computer.org>

Editorial production by Patrick Kellenberger
Cover art production by Joe Daigle/Studio Productions
Printed in the United States of America by The Printing House

IEEE Computer Society
Conference Publishing Services (CPS)
http://www.computer.org/cps

A Scientific Rapid Prototyping Model for the Haskell [illegible]

Brian J. d'Auriol, Sungyoung Lee, Young-Koo Lee
Department of Computer Engineering
Kyung Hee University
Korea
{dauriol,sylee}@oslab.khu.ac.kr,yklee@khu.ac.kr

Abstract

A simple methodology for computational and scientific rapid prototyping in Haskell is proposed. The methodology is based on identifying abstractions that are defined over entire list data structures combined with identifying a set of processes that initially construct a list data structure and subsequently modify that data structure in iterative steps leading to a final post modification process to output the data structure. A case study exemplifies and details an application of the proposed model. The results indicate that the proposed methodology can be easily implemented for developers or coders with little in-depth knowledge of functional programming capabilities; thereby, enabling applicability for a wide range of users.

1. Introduction

Nowadays, the computing infrastructure for computational sciences relies heavily on networked computers with on-demand access to high performance computers if and when needed. In addition, such computing infrastructures also include networked clusters, server-based clusters and in some cases, peer-to-peer clusters. There are many available options for the programming of these systems, depending on user choice and computing needs, for example, Java and related languages are commonly used for networked cluster based applications whereas FORTRAN and C are also commonly used for high-ended performance computations. Moreover, there are many libraries and packages that also can be incorporated into applications, for example, LAPACK, BLAS, VECLIB and MPI.

There are many computer resource, programming language, and library models that are available choices for the development and coding phases of computational applications. Many of these have individual conceptual models for procedure and data representations. In some cases, the models can integrate together; for example, typical MPI-based parallel programming often [illegible] farming (master-slave) and Single Program [illegible] (SPMD) models combined and integrated [illegible] code. Moreover, the models may have [illegible] stractions, which in some cases, may [illegible] ing conceptual layouts, for example, [illegible] well known to provide low-level point-[illegible] point communication specification, the [illegible] guage level and the latter at the library [illegible] tion. However, models based on [illegible] based languages (e.g. FORTRAN-D [illegible] guages such as Linda or Haskell [illegible] specification requirements and more [illegible] implicit' concepts. A number of [illegible] idea of program skeletons, that is, [illegible] fragments. The complexity of choice [illegible] ure 1

This paper follows from the earlier [illegible] we have defined the term: *Programming [illegible] models* to describe the viewpoint of "The [illegible] allel program is guided by at least one [illegible] guided by several." This paper continues [illegible] rection. Here, we are motivated by [illegible] choice upon the user for scientific and [illegible] model development. This scientific rapid [illegible] cess needs to be supported by model [illegible] fast development while allowing the [illegible] trate on core ideas. Hence, this paper is [illegible] the development time for research [illegible] velopers to more quickly produce [illegible] in the business, science and ubiquitous [illegible]

This rest of this paper is structured [illegible] describes the scientific rapid prototyping [illegible] requirements. Section 3 presents the [illegible] structures that we propose as suitable [illegible] tional and scientific rapid prototyping. [illegible] a case study application and conclusions [illegible] tion 5.

978-0-7695-3407-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ICCIT.2008.187

structure patterns that support rapid prototyping; in particular, that allow the developer to concentrate on *solution* as opposed to algorithm. Therefore, despite the general purpose capability and the many special functional capabilities of the language, we suggest simpler structures that we feel may be easier for the developer to program in, especially for those who do not have in-depth knowledge of functional languages. In this paper, we concentrate on the basic model of such structure patterns that is widely applicable to many programmed solutions, however, which might not be universally applicable.

Haskell's features of recursion and outer-layer pattern matching provide a nice compact semantic and grammar notation for tail recursion. Here, we introduce the term *take 1, drop 1* recursive processing of a list data structure to describe this pattern structure. There are three purposes that such a recursive procedure abstraction serves: 1) reduce a list to a scalar, 2) modify each element in a list (i.e. construct a new list), or 3) filter each element in a list (i.e. construct a sub-list). Note that any particular code fragment could combine the latter two. This structure uses the pattern specification `(x:xs)` to implement the take 1, drop 1 concept; recursion terminating upon the pattern `[]`; and explicit list construction using the `:` operator. Such a pattern requires processing individual data elements. The time required to code is comparative large.

Haskell also provides a list comprehension abstraction whereby a list construction can be specified by including three parts: the list element, the set of data inputs that collectively determine the output set and various filters over the inputs. This syntax is much more compact than the take 1, drop 1 structure, and can capture the latter two purposes of the take 1, drop 1 structure. Semantically, this structure still deals with element by element, however, in a overall single list manner.

Lastly, Haskell also provides a number of high level abstractions (and more experienced coders can implement user-defined abstractions). Examples of pre-existing functions in this category include: `fold` (and its variants), `map`, `zip` and its variants, and `filter`. Collectively such abstractions apply over an entire data structure.

Figure 3 illustrates the suggested relative abstractions of the above structures. Due to the preference of defining operations over a data structure as opposed to element by element, we determine the order of preference for coding for rapid prototyping: high level, then mid level, then low-lever. High level functions can be implemented by low level abstractions; ultimately, all code can be implemented via the recursive structure.

The semantics of using higher level functions influences the way programs are developed. This leads us to propose a simple three stage algorithm methodology: 1) create a basic data structure representing the input, modify the basic data

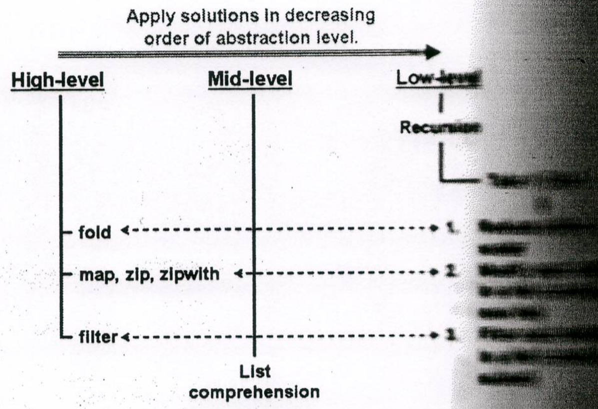

Figure 3. Abstraction levels of ide... structure patterns

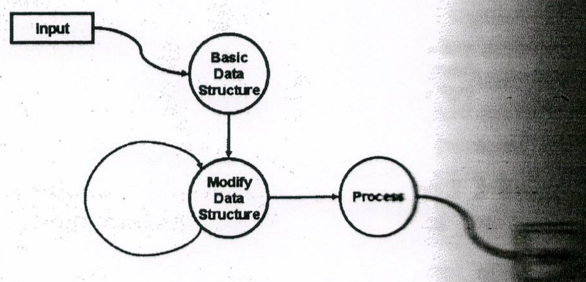

Figure 4.

structure, in some cases, by applying an ... cation steps, 3) final process of the modified ... leading to the final output. This methodology ... in Figure 4.

4 Application

We have applied the proposed model ... palindrome identification in bioinformatics. ... sults indicate the potential benefit of the ... for rapid computational and scientific ... section, we briefly show the application ... identification problem.

A palindrome in an RNA or DNA ... cleotide sequence consisting of pgp' where ... quence of length n and p' is the reverse ... also of length n and g is a subsequence ... p'. The significance of identifying such ... cleotide sequence is varied, for example, ... secondary structure formations in RNA. In ... use the proposed model to give a solution ... identification. The input sequence is ... The final output of the Haskell program ...

```
[illegible]Print 6 s
```

[illegible]	[illegible]	[illegible]ring	Second Substring		
[illegible]	[illegible]	ctggac	253	258	gtccag
[illegible]	[illegible]	caagtc	223	228	gacttg
[illegible]	[illegible]	agtccc	194	199	gggact
[illegible]	[illegible]	ttacca	135	140	tggtaa
[illegible]	[illegible]	ctgcgg	96	101	ccgcag
[illegible]	[illegible]	ctgcgg	182	187	ccgcag
[illegible]	[illegible]	tgcggc	181	186	gccgca
[illegible]	[illegible]	gcggct	180	185	agccgc
[illegible]	[illegible]	gctcgt	263	268	acgagc

[illegible] Final output of the Haskell program [illegible] on the proposed model: the report [illegible]dromes of length 6 for a portion [illegible]patitis C virus NS5 gene, sequence [illegible] from the NCBI Nucleotide database, [illegible]769711, and consists of 269 bases.

[illegible] is shown in Figure 5.

[illegible] level code is given below, in par- [illegible] functions palindromeFormat and [illegible]Header represent the final process of [illegible] methodology.

```
[illegible]Print :: Int->String->IO()

[illegible]Print n s = putStr
[illegible]dromeHeader ++
[illegible]dromeFormat (palindrome n s))
```

[illegible] palindrome contains a reference to the [illegible]uild and reverseComplement as well as [illegible]. This function is further discussed below.

```
[illegible] :: Int->String->[PalindromeInfo]
[illegible] n s =
[illegible] ls = zip (i n s) (build n s)
[illegible] ((fst x, fst x+n-1),
  (fst y, fst y+n-1), snd x, snd y) |
  x <- ls,
  y <- ls,
  x < y,
  (snd x) ==
      (reverseComplement (snd y)) ]

[illegible]
[illegible] :: Int -> String -> [Int]
[illegible] s = take (length s - n + 1) [1..]
```

[illegible] function build represents the construction of the [illegible] structure from the string representation of the [illegible] sequence. This data structure is a list of strings, [illegible] element represents a possible nucleotide subse- [illegible] the data structure contains all such prospective *p* [illegible] at successive indices in the original sequence. [illegible] that the use of the length function in the guard here is likely to be an expensive operation; for rapid prototyping, this is likely acceptable, however, subsequent refinement is needed if the prototype is to be refined for performance.

```
build :: Int -> String -> [String]
build _ [] = []
build n (x:xs)
   | (length xs) >= (n-1) =
         (x:take (n-1) xs) : build n xs
   | otherwise              = []
```

The use of the high level zip function in palindrome represents a modification to the basic data structure, hence, defining a first iteration upon the modification part of the methodology. At this point, in reference to Figure 4, this part of the process effects the transition from 'Basic Data Structure' to 'Modify Data Structure'. This modification constructs a new list with each of the prospective *ps* combined with its starting index represented as a pair (i.e., two-tuple).

The main part of palindrome is a list comprehension which implements both a modification and a filtration of the previous modified data structure. This further modifies the data structure to contain appropriate information necessary to the required output (see Figure 5) as well as filters the data structure to contain only identified palindromes (without duplication). At this point, in reference to Figure 4, this part of the process effects two transitions from 'Modify Data Structure' to 'Modify Data Structure' (i.e., there are two modifications when considered individually).

Lastly, as part of the palindrome identification filter in palindrome, the function reverseComplement is referenced which itself uses high level functions (e.g. reverse and map) to define p' as a single operation over the input subsequence string.

```
complement :: Char -> Char
complement c
   | c=='c'   = 'g'
   | c=='g'   = 'c'
   | c=='a'   = 't'
   | c=='t'   = 'a'

reverseComplement :: String -> String
reverseComplement s =
   (reverse . map complement) s
```

5 Conclusions

In this paper, we propose a simple methodology for computational and scientific rapid prototyping in Haskell. The methodology is based on identifying abstractions that are defined over entire list data structures combined with identifying a set of processes that initially construct a list data structure and subsequently modify that data structure in iterative steps leading to a final post modification process to output the data structure. We have detailed an application of the proposed model in a case study of a palindrome identification algorithm for bioinformatics. The application described here illustrates how the proposed methodology can be easily implemented for developers or coders with little in-depth knowledge of functional programming capabilities; thereby, enabling our simple approach for a wide range of users.

The approach and results in this paper, although both encouraging and preliminary, should be developed further. In particular, user experiments to establish quantitatively the benefit for rapid prototyping should be conducted. Also, we would like to more qualitatively determine the range of algorithms that can be realized by our proposed model. In line with our earlier work both in model identification of program structures and in the visualization of the same, we are motivated to re-consider our work in this paper for application in software visualization.

Acknowledgements

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement)"(IITA-2008-C1090-0801-0002) and by the MIC (Ministry of Information and Communication), Korea, Under the ITFSIP (IT Foreign Specialist Inviting Program) supervised by the IITA (Institute of Information Technology Advancement, C1012-0801-0003.softeng:Davies2005 Also, this work is financially supported by the Ministry of Education and Human Resources Development (MOE), the Ministry of Commerce, Industry and Energy (MOCIE) and the Ministry of Labor (MOLAB) through the fostering project of the Lab of Excellency.

References

[1] Haskell. http://haskell.org.

[2] P. Coveney, J. Freq, D. Gavaghan, J. Essex, and J. [illegible] Rapid prototyping of usable grid middleware. [illegible] abstract, April 2005 – April 2007.

[3] B. J. d'Auriol and J. Ulloa. Specification and [illegible] metrics for parallel programs. In H. R. Arabnia, [illegible] *Proceedings of The 2005 International Conference [illegible] ware Engineering Research and Practice (SERP'[illegible]) [illegible] Fourth International Workshop on System/Software [illegible] tures 2005 (IWSSA'05)*, pages 101–107, Monte Carlo [illegible] Las Vegas, NV, USA, June 2005. CSREA Press.

[4] N. Davies, J. Landay, S. Hudson, and A. Schmidt. [illegible] tion: Rapid prototyping for ubiquitous computing. [illegible] *vasive Computing*, 4(4):15–17, 2005.

[5] P. Hudak and M. P. Jones. Haskell vs. ada vs. c++ [illegible] ..., an experiment in software prototyping productivity. [illegible]

[6] M. Rauterberg. An iterative-cyclic software [illegible] *Proceedings of the Fourth International Conference [illegible] ware Engineering and Knowledge Engineering*, [illegible] 607, Capri, Italy, June 1992. IEEE.

[7] C. D. Rickett, S.-E. Choi, C. E. Rasmussen, and [illegible] Rapid prototyping frameworks for developing [illegible] cations: A case study. *Journal of Supercomputing*, [illegible] 134, 2006.

[8] D. Skillicorn and D. Talia. Models and languages [illegible] computation. *ACM Computing Surveys*, 30(2):123–[illegible] 1998.