DOI: 10.1515/auom-2015-0019
An. Şt. Univ. Ovidius Constanţa

VERSITA
Vol. 23(1),2015, 277–290

Upon the Haskell support for the web applications development

Anca Vasilescu

Abstract

Some of the most challenging directions in recent theoretical and practical research concern both the web science and the web applications development. Any alive science and any alive programming language should increase its connection with the web domain. Apart from general advantages provided by the functional language programming `Haskell`, its specific support for the web applications design and implementation is highlighted in this paper. It is also our target to reveal here some mathematical grounds of the web as a modern and very young science.

Appropriate examples will be selected from our students' functional web applications. So, the web ecosystem and particularly those characteristics which are relevant for guiding our students to choose `Haskell` for developing performant maths-based web solutions are pointed out. All the mathematical evaluations involved in this approach are `Haskell` based.

1 Indroduction

From the end-user point of view, probably the most challenging directions in functional programming are: parallel and concurrent programming, web application development support, generic programming or type/kind-level programming. Out of these, are the web applications the most widely used nowadays? The answer is definitively, yes, they are.

Key Words: computer science education, functional programming languages, Haskell, Page Rank metric, web applications
2010 Mathematics Subject Classification: 68N18, 97R50, 68M11, 68U35
Received: 2 May, 2014.
Revised: 20 June, 2014
Accepted: 29 June, 2014

Web applications are becoming the standard way to interact with users. The benefits of this approach are many: you do not have to create a different executable file for each system from which your client may access your application (although you need to consider interoperability issues between browsers), and you do not need users to download or execute any binary code: everything runs smoothly inside the browser [7].

It follows that, any alive programming language on the software market has to increase its support for web application development, including `Haskell`. Using the same language, for example `Haskell`, for both backend and frontend web application sides, reduces the developer effort when working on both parts at the same time, and also it favors to share the code between these parts.

Apart from general advantages provided by `Haskell`, the specific `Haskell` support for the web applications design and implementation is highlighted in this paper. It is our target to reveal the `Haskell` web ecosystem and particularly to focus on those characteristics which are relevant for guiding our students to choose `Haskell` for developing performant web applications. An appropriate example will be selected from our students' functional web applications pool, namely a personal blog. For the associated blog content, we shall analyze the significance of the internal web pages using specific metrics. We shall use the same `Haskell` programming language support for the involved computations.

As a very dynamic domain, both the developers and the researchers have to attentively look at the web context. In order to outline the sense of the modern phrase *web science* we consider here the main ideas from [2]. The general meaning of *web science* is not just about methods for modelling, analyzing and understanding the web at the various levels. It is also about engineering protocols and providing infrastructure, and ensuring that there is fit between the infrastructure and the society that hosts it. Web science is inherently interdisciplinary, and it has to be able to drive web development in socially and scientifically useful ways. The web needs to be studied and understood, and it needs to be engineered. That is why, the researchers' interest for mathematical and algorithmic foundations of the web and the internet is increasing during this decade.

Being teachers, we have also the moral duty to keep the students on the track of updated programming languages offer. Learning functional programming sets a student in a much better position as a developer, mainly because functional paradigm is closer to pass through first class programming in the near future. Hence, our students will be prepared to develop larger and faster applications that bring satisfaction and enthusiasm to the world wide customers. From this point of view, this paper aims to reveal some aspects just on the line between computer science and computer science education.

2 Theoretical aspects

2.1 Connecting functional programming and the web

By definition, `Haskell` belongs to the family of functional languages, a broad set which also includes: `ML`, `Lisp`, `Scala` or `Clojure` [7].

Notice that functional programming paradigm is increasing the programmers' interest, day by day. Popular programming languages like Java or C# are embracing the functional features, while the original languages like `Erlang`, `Haskell`, `OCaml` or `Scala` are gathering these advantages from the very beginning. Out of these examples, `Haskell` enjoys probably the greatest interest both from academic and from industrial sides. It is continuously supported by a large number of packages suppliers and a very active international events agenda.

`Haskell` is very appreciated because it supports the programmer to write correct code by at least three features of the language: (1) *strong static typing*, meaning that the type of an expression and/or the value are already known during the compilation, before the execution, (2) *pure functions*, meaning that in `Haskell` a function does not have side-effects and consequently, the code becomes predictable and easier to test and (3) *lazy evaluation*, meaning that an expression is not evaluated unless its result is required.

Following the definitions from [7], a *web framework* is a library or set of libraries prepared to be used for developing a web application. Usually, it covers all possible technological requirements, such as: a routing system - for linking a URL to a specific piece of code to be executed when this URL is to be served, a templating system - for generating the final HTML output, several kinds of caches, database access, and authentication through different protocols.

The official `Haskell` website [12] points out the most active `Haskell` web frameworks as follows: `Happstack`, `Snap` and `Yesod`. Out of these three frameworks, `Yesod` is the full-featured one because it provides solutions for templating, routing, persistence, sessions, JSON, authentication/authorization and, the most important thing, *if the code compiles then it works*. Besides, using `Yesod`, `Haskell` allows compile-time checking for correctness based on some metaprogramming facilities provided by the specialized DSLs, namely domain specific languages.

So, for `Haskell`, a specific, very good choice is `Yesod`. As part of this framework, the `Persistent` library is developed for making everything as type-safe as possible. Using `Yesod` framework, we can manage all parts of a web application, including routing and templating. For templating, different sub-libraries for handling each common output language in the web have been developed: `Hamlet` for HTML documents, `Cassius` and `Lucius` for CSS style

sheets, and `Julius` for JavaScript code.

In the recent book [9], important advantages of the `Haskell` base for web development are mentioned. Apart from giving a large performance advantage over other offers, this choice together with its related frameworks provide a specific architecture designed for performance, too. In terms of time performance, the `Yesod`'s server Warp, for example, is appreciated as the fastest `Haskell` web server around. So that, combining `Yesod` framework facilities and Warp time performance, one of the fastest web application deployment solutions is available.

In order to have a solid `Haskell` based web application, a smart haskeller could count on: simplicity of the language, simplifying the deployment for increasing performance, producing cleaner and more modular code, compiler automatically catching mistakes, type-safety and concise and declarative syntax, advanced type system features and patterns.

2.2 Connecting mathematical foundations and the web

Based on a client-server architecture, a standard web application gets a scenario like this: the clients just enter a web address into the browser and should get the content they are expecting for. The development of such an application is divided into two parts: the backend - a server that will listen for HTTP requests and return and update information, and the frontend - the code running in your clients' browsers, sending the correct requests to the backend.

An important decision to make when web applications are designed is the choice of the appropriate programming language. This step involves both the client, namely the web browser, and the server. Regarding the programming language for the server side, usually, the web developers have to choose of: dynamically typed languages like `php`, `Python`, `Ruby` or typed languages like `Java`, `C#`. However, following the new trends in programming design, there is another option that should be taken in consideration when we are speaking about the web applications: `Haskell`, the pure functional programming language.

From the end-user point of view, wikis, blogs or social networks are the most popular page types in the web framework. The internet users generate their own texts, express opinions, comment a book or a movie, post personal photos, videos, music, on their own blog or into the others' spaces. For all of them, but especially for a blog owner, some mathematical attributes should be very important for revealing the quality of his/her blog in the community. Out of these attributes, the most important state about the significance or the importance of a web page coming out from three perspectives: (a) its relevance to specific information needs, for example a user query, (b) its popularity or

the rank as a measure of its location into the specific web graph and (c) its absolute quality beyond any particular user requirements.

While the relevance of a page related to a given set of keywords is given by the presence and the positions of those keywords, the overall rank of the page depends also on its connections in the web graph. These two characteristics are largely decided through specific metrics as we shall mention bellow. But, the absolute quality of a given web page or blog content essentially depends on the user's own critical faculties, together with a content verification based on different reliable sources.

Information recovery techniques have been modified to the web area for establishing the relevance of web pages to keyword queries. There are known at least four algorithms for relevance ranking, namely *Boolean spread activation*, *most-cited*, *TF*x*IDF* for *Term Frequency combined with Inverse Document Frequency* and *vector spread activation* [5], [6].

For the interest of this paper, we consider here the operation of the last two metrics, namely *TF*x*IDF* and *vector spread activation* for deciding a blog content relevance. The notations to be used in defining the relevance metrics are: M is the number of query words, Q_j is the j^{th} query term, for $1 \leq j \leq M$, N is the number of web pages in index, P_i is the i^{th} page or its identification number, $R_{i,q}$ is the relevance score of P_i with respect to the query q, $Li_{i,k}$ is the occurrence of an incoming link from P_k to P_i, $X_{i,j}$ is the occurrence of Q_j in P_i.

Based on the vector space model, the relevance score of a blog page is the sum of weights of the query terms that appear in the document, normalized by the Euclidean vector length of the document. The weight of a term is a function of the word's occurrence frequency (also called *the term frequency*, denoted by TF) in the document and the number of documents containing the word in collection (*the inverse document frequency*, denoted by IDF). So, the relevance score of P_i with respect to the query q is defined by [5]:

$$R_{i,q} = \frac{\sum_{Q_j}(0.5 + 0.5\frac{TF_{i,j}}{TF_{i,max}})IDF_j}{\sqrt{\sum_{j \in P_i}(0.5 + 0.5\frac{TF_{i,j}}{TF_{i,max}})^2(IDF_j)^2}} \tag{1}$$

where $TF_{i,j}$ is the term frequency of Q_j in P_i, $TF_{i,max}$ is the maximum term frequency of a keyword in P_i and $IDF_j = \log(\frac{N}{\sum_{i=1}^{N} X_{i,j}})$.

The general vector space model is often avoided because it does not consider the hyperlink information as done in web page quality models. Hence, the *vector spread activation* method incorporates score propagation as done in *Boolean spread activation*. Each web page is assigned a relevance score (according to the *TF*x*IDF* model) and the score of a page is propagated to those

it refers. That is, given the score of P_i with respect to the query q as $S_{i,q}$ and the link weight $0 < \alpha < 1$,

$$R_{i,q} = S_{i,q} + \sum_{\substack{j=1 \\ j \neq i}}^{N} \alpha L i_{i,j} S_{j,q} \tag{2}$$

It follows that also the practical tests show that the *vector spread activation* metric performs only marginally better than TFxIDF in terms of retrieval effectiveness. The above table presents the average precision for each of the previous four algorithms, for a pool of fifty-six test queries [5]:

	Boolean spread activation	most-cited	TFxIDF	vector spread activation
Average precision	0.63	0.58	0.75	0.76

Concerning the second criterion, the web page popularity or authority, there are also many appropriate algorithms. The recent results in web science are proving that the popularity of a web content depends on the hyperlink structure it belongs to. Usually, the link structure analysis is based on the notion of *link* from a page p to page q, namely an endorsement of q by p. In [5] a set of largely used metrics for link structure analysis is presented, mainly the *impact factor*, the *Page Rank*, the *mutual reinforcement approach* or HITS (for *Hyperlink Induced Topic Search*) and PicASHOW.

For the interest of this paper application, a personal web blog, the Page Rank metric is presented here. This is based on the mathematical concept of a Markov chain by assigning a *popularity* score to each web page in accordance with its number of incoming links in the web (sub)graph.

The general formula for the rank $R(p)$ of the page p is as follows [6]:

$$R(p) = \sum_{q \in Q} \frac{R(q)}{L(q)} \tag{3}$$

where $R(p)$ represents the page rank of the *source* page p, Q is the set of all pages pointing to p and for each page $q \in Q$, $L(q)$ is the number of distinct pages pointed (or linked) by q. So, the rank of the page p appears as the sum of the ranks of the pages pointing to it, each divided by the number of outgoing links from those pages.

Following this definition, the Page Rank algorithm looks like helping the older pages because the new ones usually have only a few incoming links despite their actual importance. That is why, a *damping factor* d might be

added from the very beginning, considering the possibility of jumping from one node to any other node chosen randomly. The new ranking evaluation is:

$$R(p) = d * \sum_{q \in Q} \frac{R(q)}{L(q)} + \frac{1-d}{N} \tag{4}$$

where N is the total number of pages in the given blog and d is statistically appropriate to be set to 0.85. For the choice of $d = 1$ the previous formula holds and for $d = 0$ the starting proposal holds, namely all links having equal rank $\frac{1}{N}$.

The definition formulae are obviously recursive and the evaluation has to be done starting from a set of arbitrary assignments of ranks for each page. In a matrix-based representation, the previous relations (3) correspond to the next iterative system based on the specific matrix product:

$$\begin{aligned} r_1 &= L \times r_0 \\ r_2 &= L \times r_1, \text{meaning } r_2 = L^2 \times r_0 \\ &\dots \\ r_k &= L \times r_{k-1}, \text{meaning } r_k = L^k \times r_0 \\ &\dots \end{aligned} \tag{5}$$

where r_0 is the vector of initial assignments of ranks and the matrix L represents the representation given by the web graph structure and the specific model of the pages importance transfer through links. One may consider typically all equal ranks as start and make the evaluation by applying specific techniques based on: dynamical systems, linear algebra, probabilistic approach or others.

An important point is that we do not actually need to compute the limit of these rank values, so the chain of computations can be interrupted when the differences between the elements of r_k and the correspondent ones of r_{k-1} are less than a convenient value *epsilon* [6]. The sequences of iterates r_0, Lr_0, ..., $L^k r_0$ tends to the stable, equilibrium value assuming to be the Page Rank vector of the given web graph.

For our practical example, in the next section we shall use this method for evaluating the significance of our blog pages.

3 Applications

As it is mentioned in [13], the master students from the Faculty of Mathematics and Informatics, University *Transilvania* of Braşov are studying `Haskell` in the third semester. In the framework of this course, one of the most achieved

subjects for their research report is concerning the `Haskell` based web applications. For the interest of this paper, we propose one such example, namely the appropriate part from [8] as a valuable tutorial and its related application. Hence, we have an `Yesod` project, with SQLite support for creating a personal GreenBlog.

After you have created the project using *yesod init* command, you have to focus on the important *files* and *directories* of the project, as follows: (1) `config/routes`: in this file you will configure the path, namely the *routes*, from the URL address to the source code; (2) `config/models`: in this file you will configure the application persistent objects, namely database model and database tables; (3) `Handler/`: contains the *handler* files, namely the `Haskell` *.hs* files with the code called when the associated URL is accessed; in a *handler* you process user input, perform database queries and create responses; (4) `templates/`: contains users templates, namely `hamlet` files as HTML templates, `julius` files as Java Script templates, `cassius` and `lucius` files as CSS templates.

In order to see how all these work together, let us see how we can make the "About" page for our blog.

Step 1. Since we have a web application, we need some HTML, CSS or Java Script code. We shall use HTML template code and we shall put it in a `hamlet` file (the extension of the file will be `hamlet` and not `html`). We shall save the `hamlet` file in the `templates` directory. For example, the next code displays the contact details of the blog owners on the "About" page:

```
<h1>For  more information about the project, contact:
 <ul>
  <li><h2>Neagu Elena, e-mail: elena_neagu90@yahoo.com
  <li><h2>Parloaga Claudia-Stefania, e-mail: claudiaparloaga@yahoo.com
 </ul>
```

Step 2. To actually see the content of the *contact.hamlet* file described above, we need a `handler` file for connection. This `handler` is a `Haskell` file which implements the method `getContactR`. The appropriate `handler` code should be like this:

```
module Handler.Contact where
                          import Import
getContactR :: Handler Html
getContactR = do
   defaultLayout $ do
      setTitle "Contact"
      $(widgetFile "contact")
```

This code sets the title of the page using `setTitle` and calls the `hamlet` file *contact.hamlet* in the last row through a widget using the layout `defaultLayout`.

Step 3. In order to access the `handler` through an URL address, we need to define a `route`. For this, add in the file `routes` from the `config` directory the following route:

```
/contact ContactR GET
```

This line defines a route through a set of expected components: the web page URL address `http://localhost:3000/contact`, including the `handler` name `contact`, the route name `ContactR` and the type `GET` for the accepted requests from the web page. Following this definition, the route `ContactR` waits for requests from the `contact` resource and it answers the specific `GET` requests coming through the default `Yesod` port `3000`.

In order to see the page, you have to proceed in two steps: (1) start the `yesod devel` application such as the server is listening on port 3000 and (2) access the web page resource `localhost:3000/contact` from your favorite browser. You should see the content of the Figure 1.

Figure 1: <About> page

Figure 2: Blog content

Also, to set the layout for your application, you should consider to modify the `global.css` file located in the `static` directory. You can do your own layout.

Of course, using more then one `handler`, one `route` and one `hamlet` file like in the previous example, we obtain a totally functional blog. We also present in Figure 2 such a blog page from our application. Trust you: if you understand the basics, you can do any complex `Haskell` based web application by yourself!

In order to evaluate the social impact of our personal blog in the internet, we may proceed by using the specific metrics introduced in the previous section for measuring the significance of our blog web pages. For example, we shall evaluate the page ranks for our blog content in accordance with the formula (4).

Let consider the internal structure of the associated web graph given in Figure 3.

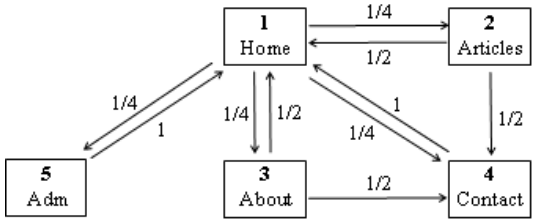

Figure 3: Blog graph

We have five pages at first level, corresponding with five nodes in this web graph. Here, a directed edge from node i to node j appears iff the web page i references the web page j. For example, page 1 <Home> links to all of the other pages, so node 1 will have outgoing edges to all of the other nodes and meantime, the page 4 <Contact> has only one link, to page 1, so that the node 4 will have one outgoing edge to node 1.

In our evaluation model, each page evenly transfers its importance to the pages that it links to. Hence, the node 1 has four outgoing edges, so it will equally pass on of its importance to each of the other four nodes. Node 4 has only one outgoing edge, so it will pass on all of its importance to node 1. Reiterating this method, we have the appropriate matrix L for our model.

We store the values of page ranks in a vector r, where $r[i]$ is the rank of page i for each $i = \overline{1,5}$. So, starting from the initial configuration of values r_0 for r, the computation is iterating as follows from the relations (5).

Suppose that initially the pages importance is equally distributed among the five nodes, each getting $\frac{1}{5}$. So, the initial rank vector is the column $r_0 = (0.2\ 0.2\ 0.2\ 0.2\ 0.2)'$. Each incoming link increases the importance of the linked web page, meaning that at each step, we update the rank of each page by adding to the current value the importance of the incoming links. This is the same as multiplying the matrix L by the current vector r step by step.

The next Haskell source code is evaluating the r components values successively.

```
rNext rPrev = [sum[fst z * snd z | z<-zip (l!!(i-1)) rPrev]|i<-[1..5]]
converge p (x:ys@(y:_))
    | p x y     = fst x
    | otherwise = converge p ys
```

```
--infinite list of pairs
--given by k and the k-th component of the successive vectors r
pairPos i = zip (iterate (+1) 0) [x !! (i-1) | x <- iterate rNext r0]
stable i eps = converge (\x y -> abs(snd x - (snd y))<eps) $pairPos i
pageRank eps = maximum [stable i eps | i <- [1..5]]
checkVal eps = [(pairPos i !! (pageRank eps - 1),
                 pairPos i !! pageRank eps,
                 pairPos i !! (pageRank eps + 1)) | i <- [1..5]]
```

The incoming rank vectors r_k, for the first six natural numbers k, are:

$$\begin{aligned}
r_0 &= (0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2)' \\
r_1 &= (0.6 \quad 0.05 \quad 0.05 \quad 0.25 \quad 0.05)' \\
r_2 &= (0.35 \quad 0.15 \quad 0.15 \quad 0.2 \quad 0.15)' \\
r_3 &= (0.5 \quad 0.875 \quad 0.875 \quad 0.2375 \quad 0.875)' \\
r_4 &= (0.4125 \quad 0.125 \quad 0.125 \quad 0.2125 \quad 0.125)' \\
r_5 &= (0.4625 \quad 0.103125 \quad 0.103125 \quad 0.228125 \quad 0.103125)'
\end{aligned}$$

Depending on the given value for *epsilon*, the corresponding equilibrium point is deduced. For example, for $eps = 0.01$ and $eps = 0.000001$ the Haskell code answers are, respectively:

```
*Main> pageRank 0.01
7
*Main> checkVal 0.01
[((6,0.43),(7,0.44),(8,0.44)),
 ((6,0.11),(7,0.10),(8,0.11)),
 ((6,0.11),(7,0.10),(8,0.11)),
 ((6,0.21),(7,0.22),(8,0.22)),
 ((6,0.11),(7,0.10),(8,0.11))]

*Main> pageRank 0.000001
22
*Main> checkVal 0.000001
[((21,0.444445),(22,0.444444),(23,0.444444)),
 ((21,0.111110),(22,0.111111),(23,0.111111)),
 ((21,0.111110),(22,0.111111),(23,0.111111)),
 ((21,0.222222),(22,0.222222),(23,0.222222)),
 ((21,0.111110),(22,0.111111),(23,0.111111))]
```

In these cases, the corresponding Page Rank values are given by the vectors $r_7 = (0.44\ 0.1\ 0.1\ 0.22\ 0.1)'$ and
$r_{22} = (0.444444\ 0.111111\ 0.111111\ 0.222222\ 0.111111)'$, respectively.

We remark the powerful Haskell support both for the web application development and for the mathematical computation involved in the Page Rank

metric evaluation. The Haskell code is robust, concise and easily understanding for mathematicians.

The art of programmer is obviously the art of combining the `Haskell` functional language facilities with the support of the external framework and packages. It is already well-known that the `Haskell` programmers are continuously interested in finding the best solutions for each problem-challenge. Fortunately, such practical examples represent significant experiences for our master course. Apart from proving our students research talent, they emphasize the `Yesod-Haskell` as a solid foundation for the web application development.

Frequently, our students consent that things with functional programming are not at all easy, especially at the very beginning. But learning `Haskell` and its related frameworks will not just add another programming language to your résumé, but will teach you a new programming style, closer to the natural language.

4 Conclusions and further work

Following this paper results, we may conclude that `Haskell` is an appropriate solution for web applications development, with solid mathematical foundations. This conclusion arises from many specific aspects, such as: the developer may count on the characteristics of `Haskell` as a functional programming language, the `Haskell` community is active and continuously provide/request new hackages for developers needs, there are many good choices for `Haskell` based web frameworks, the students' interest and the commercial users' interest for functional based applications are increasing, the web science grows up as a modern interdisciplinary science and, last but not least, the available mathematical models favor a scientific approach for modelling, analyzing and understanding the internet context.

This paper content represents our interest for functional programming support in general, and particularly for `Haskell` support in different computer science domains. Here, we have aimed the `Haskell` support for the web applications development, while our previous results from [10] or [1] are focusing on the `Haskell` support for algebraic modeling or parallel image processing, respectively.

From the educational point of view, this paper comes in the framework of our interest for permanent improving our students' professional and personal skills. Some reasons for studying `Haskell` in University have been presented by our team in [11]. It is very helpful to use the same language both for the application development and for the mathematical evaluations involved in the performance measuring process.

As further work, apart from using `Yesod`, our research may progress in the direction of using also other `Haskell` web frameworks and compare the performance of derived applications. On the other hand, we may compare the `Haskell` support and other functional language support for implementing the same real-life web application.

Finally, to mention that students are always ready to learn new programming languages because "they genuinely like to program and aren't satisfied with the languages they already know" [4] is an advantage both for our educational approach and for this paper results.

References

[1] Băicoianu, A., Pândaru, R., Vasilescu, A., *Upon the performance of a Haskell parallel implementation*, Bulletin Of The Transilvania University Of Braşov - Series III: Mathematics. Informatics. Physics, Vol **6**(**55**), No. **2** - 2013, 61-72, 2013.

[2] Berners-Lee, T., Hall, W., Hendler, J.A., OHara, K., Shadbolt, N., Weitzner, D.J., *A Framework for Web Science*, Foundations and Trends in Web Science, Vol. **1**, No **1** (2006), 1130, 2006.

[3] Bozoşan, M., *Dezvoltarea aplicaţiilor web sub Haskell*, in Romanian, Dissertation Thesis coordinated by Anca Vasilescu, Master Program in Computer Science, University *Transilvania* of Braşov, Department of Mathematics and Informatics, UTBv-MI-2013-B, 2013.

[4] Collins, G., *High Performance Web Applications in Haskell*, Track: Functional Web, QConLondon 2011, Intl Software Development Conference, March 9-11, 2011.

[5] Dhyani, D., Ng, W.K., Bhowmick, S.S., *A Survey of Web Metrics*, ACM Computing Surveys **34**, **4** (December 2002), 469-503, 2002.

[6] Luccio, F., Pagli, L., Steel, G., *Mathematical and Algorithmic Foundations of the Internet*, CRC Press, 2012.

[7] Mena, A.S., *Beginning Haskell. A Project-Based Approach*, APress, 2014.

[8] Pârloagă, C.Ş., Neagu, E., *Using Haskell for web application development*, Research Project coordinated by Anca Vasilescu, Master Program in Computer Science, University *Transilvania* of Braşov, Department of Mathematics and Informatics, UTBv-MI-2014-PN, 2014.

[9] Snoyman, M., *Developing Web Applications with Haskell and Yesod*, O'Reilly Media, Inc, 2012.

[10] Vasilescu, A., *Algebraic model for the CPU arithmetic unit behaviour*, Proc. of the Third Intl. Conf. MDIS 2011, Sibiu, Romania, October 20-12 2013, Lucian Blaga University Press 2014, Editor Dana Simian, 136-145, 2014.

[11] Vasilescu, A., Drobotă, F.R., *Reasons for studying Haskell in University*, Proc. of The 7th Intl Conf on Virtual Learning, Virtual Learning - Virtual Reality, November 2-3 2012, Braşov, Romania, Editors: Marin Vlada, Grigore Albeanu, Dorin Mircea Popovici, 394-400, 2012.

[12] ***, *The Haskell website*, http://http://www.haskell.org/haskellwiki/Haskell

[13] ***, *Haskell Communities and Activities Report*, on-line, May 2014
http://www.haskell.org/haskellwiki/Haskell_Communities_and_Activities_Report

Anca VASILESCU,
Faculty of Mathematics and Computer Science,
Transilvania University of Braşov,
Bdul Iuliu Maniu, no. 50, Braşov, Romania.
Email: vasilex@unitbv.ro