DesCo: a Web Based Information System for Descriptional Complexity Results

Nelma Moreira, Davide Nabais, Rogério Reis
{nam,dnabais,rvr}@dcc.fc.up.pt
CMUP & DCC-FC, Universidade do Porto

Technical Report Series: DCC-2011-10
Version 1.0 August 2011

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL
Tel: 220 402 900 Fax: 220 402 950
http://www.dcc.fc.up.pt/Pubs/

DesCo: a Web Based Information System for Descriptional Complexity Results

Nelma Moreira, Davide Nabais, Rogério Reis
{nam,dnabais,rvr}@dcc.fc.up.pt
CMUP, DCC-Faculdade de Ciências da Universidade do Porto
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal

August 2, 2011

Abstract

Recently the descriptional complexity of formal languages has been extensively researched. One of the most studied complexity measures for regular languages is the number of states of its minimal automaton (state complexity of the language). Other measures can be related to other structural components and other models of computation. The complexity of a language operation is the complexity of the resulting language seen as a function of the complexities of the operation arguments. This proliferous research gave origin to a multitude of results scattered over more than 200 articles, with the inevitable lack of unified terminology and notation. This makes it very difficult to a interested researcher to have a global perspective of this field and realize what is the current coverage achieved in order to know where to allocate more research efforts. In this paper we present a first step towards the development of a Web based information system were descriptional complexity results can be structurally introduced, queried, and viewed. This tool will also interact with symbolic manipulation systems in order to obtain examples and perform experimental tests. Moreover the system enables the user to easily customize the database queries in order to get novel views over the existing results and respective bounds. Here we describe the main components of the database, the technologies used and the Web user interface.

1 Introduction

Recently the descriptional complexity of formal languages has been extensively researched [10, 16, 24, 14, 3]. Descriptional complexity studies the measures of complexity of languages and operations. Usually, the descriptional complexity of an object is its shortest description which can be analyzed in the worst or average case. For each measure, it is important to know the size of the smallest representation for a given language and how the size varies when several such representations are combined or transformed. These studies are motivated by the need to have good estimates of the amount of resources required to manipulate those representations. This is crucial in new applied areas where automata and other models of computation are used, for instance, for pattern matching in bioinformatics or network security, or for model checking or security certificates in formal verification systems. In general, having succinct objects will improve our control on software, which may become smaller, more efficient and easier to certify.

Among formal languages, regular languages are fundamental structures in Computer Science. One of the most studied complexity measures for regular languages is the number

of states of its minimal deterministic finite automaton (state complexity of the language). The state complexity of an operation over languages is the complexity of the resulting language as a function of the complexities of its arguments. Both concepts can be extended to other models of computation (e.g. nondeterministic automata, two-way automata, regular expressions, grammars, etc.), other measures (number of transitions, number of symbols, etc.) and other classes of languages (classes of sub-regular languages, context-free languages, recursive languages, etc). Knowing the descriptional complexity and succinctness of the objects has also obvious consequences for the computational complexity of the algorithms that manipulate them. Unfortunately, a multitude of results are scattered over more than 200 articles, with the inevitable lack of unified terminology and notation. This makes it very difficult to have a global perspective of this field and realize what is the current coverage achieved in order to know where to allocate more research efforts. All these different aspects and the huge number of results obtained, mainly in the last decades, motivates the need of a tool that helps to structurally organize, visualize and manipulate this information.

In this paper we present a first step towards the development of a Web based information system for descriptional complexity results. This is not an easy task as most of the concepts involved are of mathematical nature and thus of difficult classification. As related work we can cite a few systems that deal with (somehow) similar data but with different aims and solutions. Neil Sloane's *The On-Line Encyclopedia of Integer Sequences* [8] collects about 200,000 sequences of numbers which first collection was published on book in 1970's [23]. For each integer sequence, either only a few terms are known or a closed or recursive formula is given. In the Web site it is possible to search a sequence given some initial terms, or its closed formula or what combinatorial objects it enumerates, etc. A smaller and more recent project is the *The Encyclopedia of Combinatorial Structures* [20], that is also available as a symbolic algebraic manipulation package. Here each integer sequence is associated with a decomposable combinatorial structure making it possible to automatically compute several properties such us generating functions, closed formulas, asymptotic estimates, etc. The same Web site includes a *Dictionary of Mathematical Functions* [19]. The *Complexity Zoo* [1] is a Wiki that contains information about computational complexity classes and related topics. For each class, there is a textual description of it, problems that are known to belong to that class, and relations with other classes. More sophisticated but also more restricted is *The Navigator on Description Logic Complexity* [26] where results on the computational complexity of reasoning in Description Logics can be browsed.

The rest of this paper is organized as follows. In Section 2 we describe the information kept in the database, while introducing key notions used in the system. In Section 3 we describe the main components used in the DesCo system. Section 4 summarizes the interface's structure and functionality. Finally, in Section 5 we comment in some future work.

2 Descriptional Complexity Database

In this section we describe the information that is kept in the database, while introducing some key notions used in the system. A formal language is a set of words over a given alphabet (finite set of symbols). A class of languages is just a set of languages. A model of computation can recognize or represent a class of languages. For instance, regular languages are recognized by finite automata or represented by regular expressions. A class can also be defined by a set of basic languages and its closure under a set of operations. For more details

on formal languages and models of computation we refer the reader to Hopcroft *et al.* [15]. The most important objects that the our database stores are:

- Language Classes
- Models of Computation
- Operations
- Language Families
- Complexity Measures
- Complexity Bounds

Along with the basic information, some additional details, that will be useful in the future, for interacting with the FAdo system [21], such as which the FAdo method generates a deterministic automaton for a given language family, or performs a operation between given languages. Connections with other symbolic manipulation systems can also be incorporated.

The database was designed with flexibility in mind. For example, if someone obtains a result in descriptional complexity using a non-classical computational model, this new model can be easily added, and so can the new result. Moreover newer results do not replace older ones, instead they are just given more relevance if they are considered "better".

We briefly describe each of the above mentioned objects.

Language Classes — A language class is defined with a name and a description. Additional information can include a model of computation that characterize this class (if known).

Models of Computation — Since the descriptional complexity of operations varies with each computational model, it makes sense to store information about them so that we can define the type of both the arguments and the result of each operation. For each computational model, the database stores its name, abbreviation, a description and which FAdo's object can be used to represent it. The description can store a wide variety of information, such as a mathematical description or certain properties that hold for a specific model. A deterministic finite automaton, for example, would have DFA, as its abbreviation, `FAdo.fa.DFA` as its FAdo object, and could be described as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ consisting of:

- a finite set of states Q
- a finite set of input symbols Σ, called the alphabet
- a transition function $\delta : Q \times \Sigma \to Q$:
- a initial state $q_0 \in Q$
- a set of final states $F \subseteq Q$

Operations — Operations on formal languages include all boolean operations usually defined on sets plus other specific operations. For instance, concatenation of two languages L_1 and L_2, is defined by $AB = \{w_1w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$. The Kleene star (or iteration) of a language L is defined by $L^\star = \bigcup_{n\geq 0} L^n$, where L^n is the concatenation of L n times.

Associated to an operation we have, a name, a symbol (represented in LaTeX), a description, its arity, if its a combined operation or not , and, in the case of being a combined operation, which operations it is composed of. The purpose of the LaTeX symbol is mainly for displaying more intuitive information than just the name of the operation, but also an attempt to standardize symbols used in the literature. The description field describes the operation as we illustrated above.

To represent combined operations, we chose to use a tree structure, very much like how regular expressions are represented in FAdo. Suppose we want to define the following operation: $O = L_1 \cup L_2^\star$ (union of Kleene star). The operation O can be defined considering that the first operation is $\cup$ (union), the second on is * (Kleene star) and that it is applied to the second argument of the first operation. Every combined operation can be defined recursively in this manner.

Language Families — To show that a certain complexity bound can be reached, examples of language families L_n, where n is related to a complexity measure, must be given. These language families can be described by a indexed formula (i.e by extension), like $L_n = \{x \in \{a, b\} \mid \#_a(x) = 0 \mod n\}$). These languages can also be defined by models of computation that recognize them, for instance L_n can be defined by the family of minimal DFAs represented in Fig. 1.

Along with the actual representation, information about its format is kept. For example, if it is a diagram described in Vaucanson-G, a LaTeX package for drawing automata diagrams [18], that description can be used to generate an image file and be stored as such.

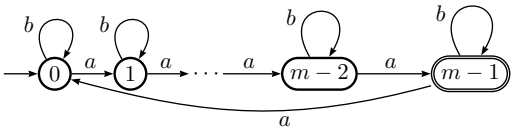

Figure 1: A family of minimal DFAs recognizing L_n

Complexity Measures — A complexity measure of a language is a function of the size of an associated model of computation. For complexity measures we keep a name and a description. The name, for example, could be *state complexity* and its description, "*The state complexity of a regular language L is the number of states of its minimal DFA* ".

Complexity Bounds — Here we focus on the complexity of operations on languages. As said before the complexity of an operation for a given measure and a given kind, is a function of the complexity of its arguments. For instance the *state complexity of a binary operation* $\circ$ *on regular languages* can be stated as following decision problem:

- Given an m-state DFA A_1 and an n-state DFA A_2.

- How many states are sufficient and necessary, in the worst case, to accept the language $L(A_1) \circ L(A_2)$ by a DFA?

Normally, an upper bound is obtained by providing an algorithm that, given the minimal DFAs for the operands, constructs a minimal DFA that accepts the resulting language. The number of states of this minimal DFA (as function of the state complexities of the operands) is an upper bound for the state complexity of the referred operation. To show that an upper bound is tight, for each operand a family of languages (one language, for each possible value of the state complexity) must be given such that the resulting automata achieve that bound.

The same approach is used to obtain other complexity bounds. Thus, in general, we consider the following fields:

Operation: the operation that we are considering;

Complexity kind: if its worst-case or average-case;

Complexity measure: as defined above, e.g *state complexity*; it has associated a parameter for each of the operation arguments;

Arguments and result types: which computational models are used and which language classes the arguments must belong to;

Complexity function: the operation complexity as function of the arguments complexity;

An algorithm: that for a given model of computation calculates the correspondent model of the resulting language having as input models for the argument languages;

Alphabet size: The complexity bounds (and witnesses) can depend on the size of the languages alphabet; this size is the smaller size for which the bound applies;

Tightness: whether the upper bound is reachable or not;

Restrictions on argument parameters: For which values of the parameters this bound apply;

Witnesses: For each argument (operand) a family of languages that ensures that the upper bound is reached

References to the literature: The references store a BibTeX entry, a title, an author and a year for articles on which the result can be found. Note that this BibTeX entry also contains the title, authors, and year of the article, however by having this redundancy of information, searching and displaying are made a lot easier.

Example 1 *Here is an example of the information of a tight bound of the state complexity of the union of two regular languages.*

Operation: Union

Complexity kind: Worst-case

Complexity measure: State complexity

Arguments and result types: Regular and DFA

Complexity function: mn

Restrictions on argument parameters: $m \geq 1$, $n \geq 1$

An algorithm: Product Construction

Alphabet size: $k = 2$

Tightness: Yes

Witnesses: $\Sigma = \{a, b\}$, $m \geq 1$, $n \geq 1$

$$L_1 = \Sigma^\star - \{x \in \{a, b\} \mid \#_a(x) = 0 \mod m\},$$
$$L_2 = \Sigma^\star - \{x \in \{a, b\} \mid \#_b(x) = 0 \mod n\}$$

References: [25]

3 DesCo Components

The DesCo system is implemented in Python [7], and has three main components: a Web framework, an interface to relational databases and a mathematical typesetter. In this section, we describe each of these components and briefly explain how they can be used.

3.1 Web Framework

Pylons [9] is an open source Web application framework written in Python, that makes extensive use of the Web Server Gateway Interface [6] standard to promote re-usability and to separate functionality into distinct modules. In the past, developers typically wrote web applications as a series of simple CGI scripts, each of which would be responsible for accessing the database and generating HTML to produce the pages it output. Although each individual script was quick to write and easy to understand, there are a few disadvantages with this approach. Every script in the site needs the same code to load configurations and to handle errors. CGI scripts can quite slow, since the whole Python interpreter, as well as, the modules the scripts uses have to be loaded into memory on each request. Also, it can be difficult to understand how the application is structured because each script is almost autonomous. To address these problems, Pylons (as well as other popular frameworks such as Django [17], TurboGears [5] and Ruby on Rails [12]) use a Model View Controller (MVC) architecture.

The MVC Architecture is a result of the recognition that most web applications:

- Store and retrieve data (the model)
- Represent data, usually as HTML pages (the view)
- Execute code to manipulate the data and control how it is interacted with (the controllers)

In Pylons each of these components is kept separate. Requests are directed to a controller, which is a Python class which handles the application logic. The controller then interacts with the model classes to fetch data from the database. Once all the information has been gathered, the controller passes this information to a view template where an HTML representation of the data is generated and sent to the user's browser.

3.2 Database Interaction

SQLAlchemy [4] is a Python library created to provide a high level interface to relational databases such as PostgreSQL [11], MySQL [2] and SQLite [13]. It allows the mapping of Python objects to database tables. For instance, consider the following (MySQL output) table for models of computation:

```
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| abbreviation | varchar(10) | NO   | PRI |         |       |
| name         | varchar(80) | YES  |     | NULL    |       |
| description  | text        | YES  |     | NULL    |       |
| FAdoClass    | varchar(80) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
```

This table can be mapped to the following Python object:

```
class Model(object):
    def __init__(self, abbreviation, name,
                 description, FAdoClass):
        self.abbreviation = abbreviation
        self.name = name
        self.description = description
        self.FAdoClass = FAdoClass
```

This approach is very convenient, since every object instance corresponds to a table record, and this object can have any number of methods implemented, it becomes very easy to manipulate table records. Using SQLAlchemy also has the advantage of not binding the applications source code to a particular database. If we decide that MySQL does not suit our needs, changing to PostgreSQL, for example, only requires that we tell SQLAlchemy to use a different DB-API driver.

3.3 Mathematical Typesetting

MathJax [22] is a collection of JavaScript programs and support files for displaying mathematics in HTML pages. It works on all modern browsers and does not require any special downloads by the end-user (unlike MathML). It uses HTML/CSS and unicode fonts for high quality typesetting that is scalable and prints at full resolution. MathJax can display mathematical notation written in LaTeX or MathML markup. However, since it is only meant for math display, only the subset of LaTeX used to describe mathematical notation is supported. One of its most notable features is its ease of use. Say you want to display the golden ratio equality on a web page. The page source only needs to load MathJax and have the equality between $$. The following code generates a page, which is exhibited in Fig. 2:

```
<!-- HMTL for golden ratio -->
<script
 src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=default">
</script>
$$ \Phi = \frac{1 + \sqrt{5}}{2} $$
```

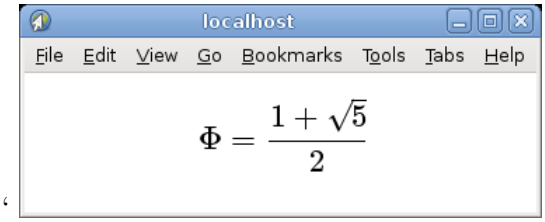

Figure 2: Epiphany web browser displaying a MathJax generated formula

4 Web Interface

The Web interface allows the manipulation and visualization of all the existing results and respective bounds, as well as, all the objects described in Section 2. Its source code is generated using Mako, a Python templating language. Its structure and functionality will be summarized in this section.

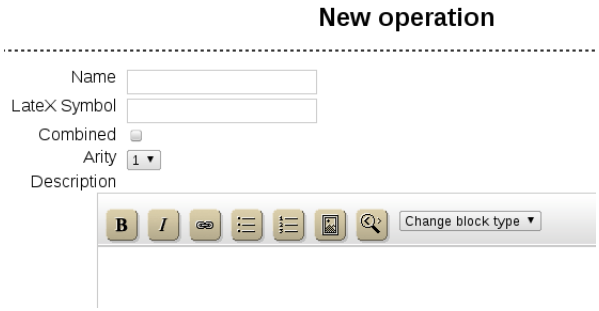

Figure 3: Form for adding a new operation

4.1 Inserting and Updating Data

Inserting and updating data through the web interface is done using a series of forms built with common HTML input elements, such as, text boxes, check boxes, radio buttons, etc, and rich text editors. In Fig. 3 is shown the form for adding a new operation. Such tasks can only be performed by authenticated users, and the information submitted is subject to review and approval by a moderator.

4.2 Interrogating and Consulting

Querying the database, in contrast to inserting and updating data, does not require any kind of authentication, thus the information is available to everyone. Querying the database on language classes, language families, operations, models of computation, complexity kinds or complexity measures results in a listing of the table contents. This listing is formatted using HTML and MathJax for a more aesthetically pleasing visualization. An example of such a listing is illustrated in Fig. 4.

Cyclic Shift	L^{CS}	1	The *cyclic shift* of a language L is defined as $L^{CS} = \{vu \mid uv \in L\}$
Plus	L^{+}	1	$L^{+} = L^{\star} - \{\varepsilon\}$
Star	$L^{\star}$	1	The *Kleene Star (also know as Kleene operator or Kleene closure)* of a language $\mathcal{L}$ can be defined as: • $\mathcal{L}^{\star} = \{x_1 x_2 \ldots x_n \mid n \geq 0$ and $x_i \in \mathcal{L}, 1 \leq i \leq n\}$ The following properties hold for the Kleene Star operation: $A^{\star}A^{\star} = A^{\star}$ $(A^{\star})^{\star} = A^{\star}$

Figure 4: Listing operations

The complexity bounds can be queried in a more dynamic fashion, giving control over how the information is displayed by customizing query parameters. Firstly, one must choose the complexity kind and measure to use. Secondly, one can choose to query all the results for a single operation, generating a table with results for all language classes and all models of computation for that specific operation, like in Fig. 5.

	Worst-Case Upper Bound State Complexity for Union		
Language	**SC**	$\lvert\Sigma\rvert$	**NSC**
Regular	mn	2	
Finite	$mn-(m+n)$	$2+(m-2)(n-2)-1$	$m+n-2$, if $m \geq 2, n \geq 2$

Figure 5: Listing results for union operation

On the other hand, one can also choose a list of operations and a specific model of computation (Fig. 6), generating a table for each language class with all the results for the chosen operations, presented in Fig. 7.

5 Conclusion

DesCo is currently available online (`khilas.dcc.fc.up.pt/desco`) and an authenticated user can also submit new information. Most of the data accessible is related with the operational complexity of regular or subregular languages. Even for this subset of descriptional complexity results it is now much easier to have a general overview, analyse the complexity behaviour of different operations for a language class, or compare the relative complexity of the same operation on different language classes. We can consider several directions of future work. There are several hierarchies of formal languages that have been studied, thus the database should include information pertaining to those hierarchies and an inference system able to reason over the available data. The interface of the database with the FAdo system (or other symbolic manipulation tool for formal languages) will be very useful for research, as almost all witnesses lower bounds are first obtained by experimental testing. The contributions of the community for further improvements and submissions will be crucial for

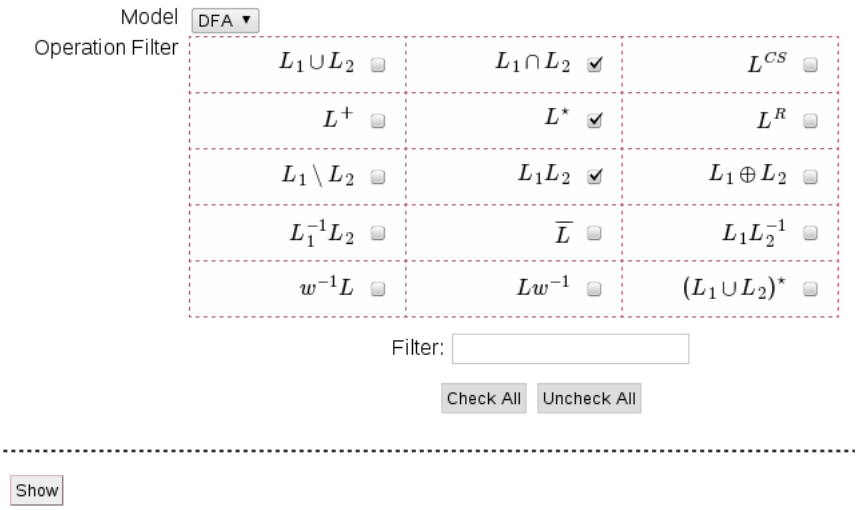

Figure 6: Selecting a model of computation and operations

the success of this project.

References

[1] Scott Aaronson. The Complexity Zoo, 2011. Available from: `qwiki.stanford.edu/index.php/Complexity\_Zoo`.

[2] MySQL AB. MySQL, 2011. Available from: `www.mysql.com`.

[3] Janusz A. Brzozowski. Quotient complexity of regular languages. *Journal of Automata, Languages and Combinatorics*, 15(1/2):71–89, 2010.

[4] Rick Copeland. *Essential SQLAlchemy*. O'Reilly Media, 2008.

[5] Kevin Dangoor and Mark Ramm. TurboGears, 2011. Available from: `turbogears.org`.

[6] P. J. Eby. Python web server gateway interface v1.0.1, 2011. Available from: `www.python.org/dev/peps/pep-3333`.

[7] Python Software Foundation. Python programming language, 2011. Available from: `www.python.org`.

[8] The OEIS Foundation. The On-line Encyclopedia of Integer Sequences (OEIS), 2011. Available from: `oeis.org`.

[9] James Gardner. Pylons book, 2011. Available from: `pylonsbook.com`.

[10] Jonathan Goldstine, Martin Kappes, Chandra M. R. Kintala, Hing Leung, Andreas Malcher, and Detlef Wotschke. Descriptional complexity of machines with limited resources. *J. UCS*, 8(2):193–234, 2002.

[11] PostgreSQL Global Development Group. PostgreSQL, 2011. Available from: `www.postgreSQL.org`.

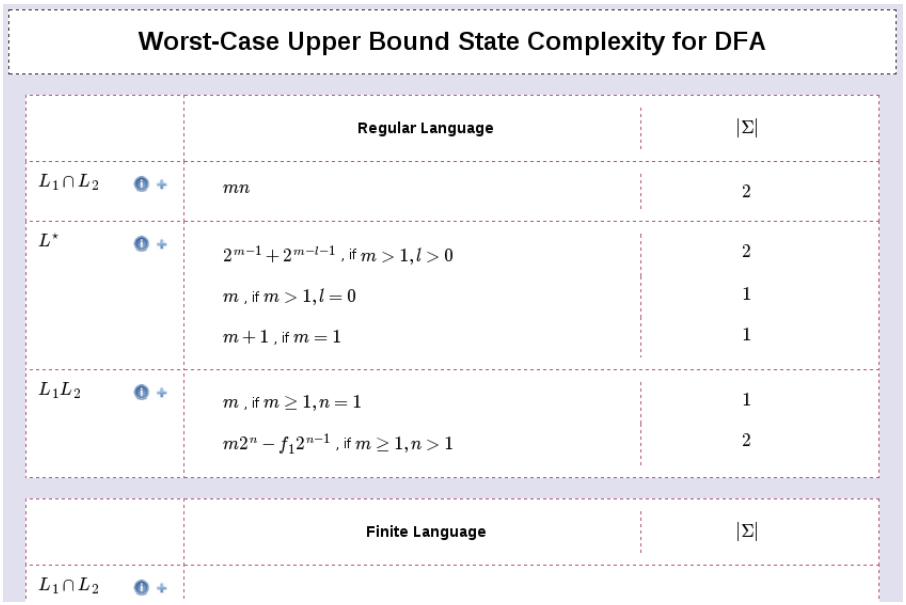

Figure 7: Listing results for several operations and language classes

[12] David Heinemeier Hansson and Rails Core Team. Ruby on Rails, 2011. Available from: `rubyonrails.org`.

[13] D. Richard Hipp. SQLite, 2011. Available from: `www.sqlite.org`.

[14] Markus Holzer and Martin Kutrib. Descriptional and computational complexity of finite automata - a survey. *Inf. Comput.*, 209(3):456–470, 2011.

[15] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 2006.

[16] Juraj Hromkovic. Descriptional complexity of finite automata: Concepts and open problems. *Journal of Automata, Languages and Combinatorics*, 7(4):519–531, 2002.

[17] Lawrence Journal-World. Django, 2011. Available from: `www.djangoproject.com`.

[18] Sylvain Lombardy and Jacques Sakarovitch. VauCanSon-G: A package for drawing automata and graphs, 2011. Available from: `www-igm.univ-mlv.fr/~lombardy/Vaucanson-G/`.

[19] Algorithms Project. Dynamic Dictionary of Mathematical Functions (DDMF), 2011. Available from: `ddmf.msr-inria.inria.fr/1.6/ddmf`.

[20] Algorithms Project. Encyclopedia of Combinatorial Structures (ECS), 2011. Available from: `algo.inria.fr/encyclopedia/intro.html`.

[21] FAdo Project. FAdo: Tools for formal languages manipulation, 2011. Available from: `fado.dcc.fc.up.pt`.

[22] Design Science. MathJax, 2011. Available from: `www.mathjax.org`.

[23] N. J. A. Sloane and S. Plouffe. *The Encyclopedia of Integer Sequences*. Academic Press, 1995.

[24] Sheng Yu. State complexity: Recent results and open problems. *Fundam. Inform.*, 64(1-4):471–480, 2005.

[25] Sheng Yu, Qingyu Zhuang, and Kai Salomaa. The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.*, 125(2):315–328, 1994.

[26] Evgeny Zolin. Navigator on description logic complexity, 2011. Available from: `www.cs.man.ac.uk/~ezolin/dl/`.