DesCo: a Knowledge Based System for Descriptional Complexity of Formal Languages

Nelma Moreira Davide Nabais Rogério Reis
CMUP & DCC, Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 4169-007 Porto, Portugal

Technical Report Series: DCC-2013-12
Version 1.0 July 2013

Departamento de Ciência de Computadores
&
Laboratório de Inteligência Artificial e 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 Knowledge Based System for Descriptional Complexity of Formal Languages*

Nelma Moreira, Davide Nabais, Rogério Reis
{nam,dnabais,rvr}@dcc.fc.up.pt

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 a few hundred articles, with the inevitable lack of unified terminology and notation. This makes it very difficult for an 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 knowledge base and a Web interface where descriptional complexity results can be structurally introduced, queried, and viewed. We also show how the system can interact with formal language symbolic manipulators in order to obtain examples and perform experimental tests. Moreover the system enables the user to easily customize queries in order to get novel views over the existing results.

1 Introduction

Recently, the descriptional complexity of formal languages has been extensively researched [9, 12, 19, 10, 5]. Descriptional complexity studies the measures of complexity of languages and operations. Usually, the descriptional complexity of an object is the size of its shortest description which can be considered in the worst or average case. For each measure, it is important to know the size of the smallest representation for a given language as well as 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. Despite their apparent weak expressive power (lowest level of Chomsky hierarchy), regular languages have applications in almost all areas of computer science. The general decidability of their properties and operations, and in many cases the linear or low polynomial computational complexity of the available algorithms, is also an attractive property for this class of languages. In particular, when compared with the undecidability world of context-free languages, just in the next level of Chomsky hierarchy. So being, it is essential that the structural properties of regular language representations are deeper researched. 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

*This work was partially funded by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT under projects PEst-C/MAT/UI0144/2011 and CANTE-PTDC/EIA-CCO/101904/2008.

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. This proliferous research gave origin to a multitude of results that are scattered over a few hundred 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 couple of decades, motivates the need of a tool that helps to structurally organize, visualize and manipulate this information. In this way, researchers and software engineers working in applications based on automata and formal languages can also more easily have access to information that can help to improve the performance of their algorithms.

In this work we present a first step towards the development of a Web based knowledge system for descriptional complexity results. This is not an easy task as most of the concepts are abstract and difficult to classify and instantiate. For instance, which are the main concepts to describe the complexity of a language operation; which is the best way to represent parameterized families of languages and how to determine if two different representations correspond to the same family; etc..

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* [18] collects about 200,000 sequences of numbers the first collection of which was published as a book in 1970's [17]. 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* [3], 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 as generating functions, closed formulas, asymptotic estimates, etc. The same Web site includes a *Dictionary of Mathematical Functions* [2]. 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* [21] where results on the computational complexity of reasoning in Description Logics can be browsed. More recently, *minicomplexity* [13] website was created, which focuses on the complexity of two-way finite automata.

The rest of this paper is organized as follows. We start off by giving some simple examples, about descriptional complexity, in Section 2. In Section 3 we describe the information represented in the knowledge base. Section 4 describes how the DesCo typesets mathematical notation. Section 5 summarizes the system functionality in terms of how the knowledge base can be interrogated and consulted. In Section 6, we explain how the FAdo system [7], a symbolic manipulator of formal languages, is used to aid DesCo in generating language families and to perform experimental tests. Finally, in Section 7 we comment in some future work.

2 Examples of Descriptional Complexity

This section, aims to help the reader become familiarized with some of the concepts we want to manipulate, by giving two simple examples of descriptional complexity of regular languages. Consider the following language, over the unary alphabet $\Sigma = \{a\}$, $L = \{a^{2n} \mid n \geq 0\}$ (or, in other words, $L = \{\epsilon, aa, aaaa, aaaaaa, ...\}$, where ϵ represents the empty-word). Since the language L is regular, it can be recognized by a deterministic finite automaton (DFA). In fact, it can be recognized by infinitely many DFAs. To illustrate this fact, two examples of DFAs that recognize L are given in Fig. 1. Since DFA B is the (state) minimal DFA that recognizes L, we say that the descriptional complexity, in

respect to the number of states, i.e., the state complexity, of L, denoted by $sc(L)$, is 2.

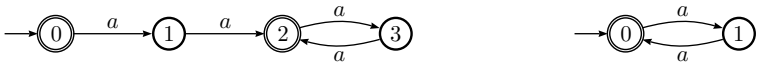

Figure 1: Both DFA A (left) and DFA B (right) recognize the language L

Let us look at another example. In Fig. 2, we present a nondeterministic finite automaton (NFA) C with three states. It is well known that any n-state NFA, can be converted, via *subset construction*, to an equivalent DFA, i.e. that recognize the same language, with at most 2^n states. This conversion is called *determinization*. If we apply the *subset construction* to NFA C, we will obtain the DFA shown in Fig. 2, which is minimal, and thus we can conclude that the state complexity of the language recognized by NFA C, or $sc(\mathcal{L}(C))$, is $2^3 = 8$, whereas the non-deterministic state complexity of the same language, $nsc(\mathcal{L}(C))$, is 3.

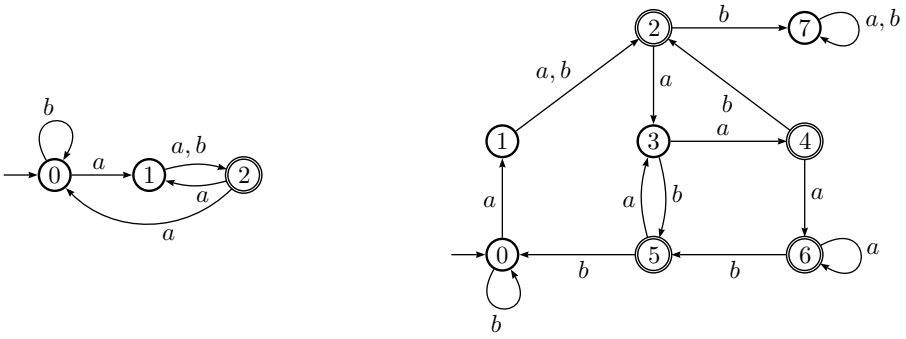

Figure 2: NFA C (left) and its equivalent minimal DFA (right) with 2^3 states

3 Descriptional Complexity Knowledge Base

In this section we describe how the information is represented in the knowledge base. We begin by giving some notions of formal languages. An *alphabet* Σ is a finite set of symbols. A *word* is a sequence of symbols from an alphabet. The set of all words A *formal language* is a set of words over a given alphabet. The set of all words over Σ is denoted by $\Sigma^\star$. A *class of languages* is just a set of languages with same properties in common. 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. Examples of models of computation are DFA, NFA, grammar, regular expression, Turing Machines, etc. Each model is defined by several sets of objects, such as, states, transitions, operators, etc. A class can also be defined by a set of basic languages and its closure under a set of language operations. Given a language $\mathcal{L}$, a model of computation $\mathcal{M}$ and a measure of $\mathcal{M}$ (e.g. number of states/transitions/symbols), the descriptional complexity of the language $\mathcal{L}$ is the minimum size of a model $\mathcal{M}$, in respect to the given measure, that recognizes or accepts $\mathcal{L}$. For more details on formal languages and models of computation we refer the reader to Hopcroft *et al.* [11].

The most important concepts we are going to consider are: *Language Classes*, *Models of Computation*, *Language Operations*, *Language Families*, *Complexity Measures* and *Operational Complexities*. Along with the basic information, some additional details, that will be useful in the future, are also taken into consideration. For example, data for interacting with the FAdo system, such as which FAdo method generates a deterministic automaton for a given language family, or performs an operation between some given languages. Connections with other symbolic manipulation systems can also be incorporated.

We briefly describe each of the above mentioned concepts.

3.1 Language Classes

A language class is defined with a name and a description. Given a preorder on language classes, a hierarchy can be considered. Currently, we define an inclusion hierarchy. In addition, witnesses of non-emptiness of language classes are available.

The class of regular languages, for instance, can be defined recursively, over an alphabet Σ, as follows:

- The empty language $\emptyset$ is regular
- The empty string language $\{\varepsilon\}$ is regular
- For each $a \in \Sigma$, the language $\{a\}$ is regular
- If A and B are both regular languages, then $A \cup B$, $A \cap B$, AB and $A^\star$ are regular languages
- No other languages over Σ are regular

3.2 Models of Computation

Models of computation are represented by a name, an abbreviation, a description and the FAdo's class that corresponds to 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 class, 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 $F \subseteq Q$

3.3 Complexity Measures

A complexity measure of a language is a function of a size of an associated model of computation. Complexity measures are described by a name, a description and the associated model of computation. The name, for example, could be *state complexity* and its description, "*The state complexity of a regular language L, denoted by $sc(L)$, is the number of states of its minimal DFA*".

Complexity measures are usually referenced, in descriptional complexity results, by a variable, instead of their shorthand notation. For example, state complexity is typically referred to by n or m, instead of $sc(L)$. For this reason, we choose to include these commonly used variables in the way we describe complexity measures.

3.4 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 $L_1L_2 = \{w_1w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$. We also consider the simulation (conversion) of different models of the same language. To characterize an operation we use a name, a symbol (represented in LaTeX), a description (as illustrated above), its arity, if it is 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 a more intuitive label than just the name of the operation, but also an attempt to standardize symbols used in the literature.

3.5 Operational Complexities

The descriptional complexity of an operation over a class of languages, for a given measure and a given model, is the (worst-case) complexity of a language resulting from the operation, considered as a function of the descriptional 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?

To obtain an upper bound, usually, an algorithm is provided, such that, given DFAs as the operands, constructs a DFA that accepts the resulting language. The number of states of this DFA (as a 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, a family of languages (one language, for each possible value of the state complexity) can be given, for each operand, such that the minimal automata resulting from the operation (i.e the state complexity of resulting language) achieve that bound, if not, then they at least provide a lower bound.

The same approach can be used to obtain other operational complexities. To specify an operational complexity, the following information is considered:

Operation: the operation that we are considering;

Complexity kind: if it is worst-case or average-case;

Complexity measure: as defined above, e.g. *state complexity*;

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

Result type: what model represents the resulting language;

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 operational complexity (and witnesses) can depend on the size of the languages alphabet;

Tightness: whether the complexity function is reachable or not;

Restrictions on argument parameters: For which values of the parameters the complexity function applies (e.g. is an upper bound);

Witnesses: For each argument (operand), a family of languages that ensures that the upper bound is reached (or provides a lower bound);

References to the literature: The references store a BibTeX entry, for articles on which the result can be found.

An illustrative example, of worst-case state complexity, follows.

Operation: Catenation

Argument types: Both arguments must be DFAs over regular languages

Result type: The result is also a DFA

Complexity function: $m2^n - f_1 2^{n-1}$

Algorithm: An adaptation of the usual algorithm for catenation of NFAs followed by a specialized subset construction.

Alphabet size: Must be greater than one

Restrictions on argument parameters: $m \geq 1, n > 1, f_1 \geq 1$, where:

- f_1 is the number of final states of the first argument
- m is the number of states of the first argument
- n is the number of states of the second argument

Witnesses: The following language families ensure the complexity function is reachable:

- For the first argument, $A = ([0, m-1], \{a, b, c\}, \delta, 0, \{m-1\})$, and for all $i \in [0, m-1]$,

$$\delta(i, X) = \begin{cases} (i+1) \mod m, & \text{if X=a} \\ 0, & \text{if X=b} \\ i, & \text{if X=c} \end{cases}$$

- For the second argument, $B = ([0, n-1], \{a, b, c\}, \delta, 0, \{n-1\})$, and for all $i \in [0, n-1]$,

$$\delta(i, X) = \begin{cases} i, & \text{if X=a} \\ (i+1) \mod n, & \text{if X=b} \\ 1, & \text{if X=c} \end{cases}$$

References to the literature: [20]

3.6 Language Families

To show that a certain operational 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 extensionally by a parameterized condition, such as $L_m = \{x \in \{a, b\} \mid \#_a(x) = 0 \mod m\}$. These languages can also be defined by models of computation that recognize them, for instance L_m can be defined by the family of minimal DFAs represented in Fig. 3, or by the following regular expression, $(b^\star + (a(b^\star a)^{m-1}))^\star a(b^\star a)^{m-2}b^\star$.

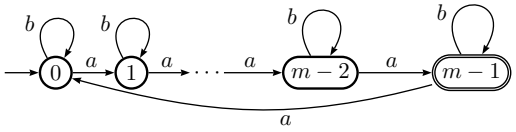

Figure 3: A family of minimal DFAs recognizing L_m

It is also possible to represent DFAs using *transformations* [14]. A *transformation* of a set Q is a mapping of Q into itself. In general, a transformation has the form:

$$t = \begin{pmatrix} 0 & 1 & \dots & n-2 & n-1 \\ s_0 & s_1 & \dots & s_{n-2} & s_{n-1} \end{pmatrix}$$

Each DFA naturally induces, for each symbol of the alphabet, a transformation on its set of states. Without loss of generality, we assume that the set of states $Q = \{0, 1, \dots, n-1\}$, and that the alphabet $\Sigma = \{0, 1, \dots, k-1\}$. We can describe a DFA by the transformations induced on its set of states, its initial state and set of final states. For example, the DFA $\mathcal{A}$, presented in Fig. 4, can be described by its initial state 0, its set of final states $\{2, 3\}$, and its transformations t_0 and t_1:

$$t_0 = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix}, t_1 = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 2 \end{pmatrix}$$

Note that the "first" line of each transformation is always the sequence $0, \dots, n-1$, so it can be omitted, and thus t_0 becomes (1 0 1) and t_2 becomes (2 2 2).

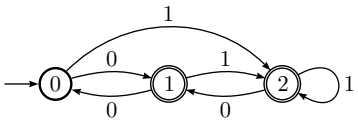

Figure 4: DFA $\mathcal{A}$

This concept can be used to specify families of DFAs in a clean and compact way. The family L_m, for example, is represented in the following way:

$$m; m; 2; 0; [m-1]; [1..m-1, 0]; [0..m-1]$$

The first field, m, is the families parameter. The second field, m, is the number of states each instance must have. The third and fourth fields are the alphabet size and initial state, respectively, followed by a list of final states, and finally, a list of transformations for each symbol of the alphabet. In this case, since $|\Sigma| = 2$, we have two lists, that describe the transition function. This first transformation states the following:

$$\delta(i, 0) = i + 1 \mod m, \forall i \in Q$$

And the second transformation describes the identity function, i.e.,

$$\delta(i, 1) = i, \forall i \in Q$$

In general, a family of DFAs, can be described using the following syntax:

$$p; n; k; i; F; T$$

where,

- p is the families parameter, that can be used in other components,
- n is the number of states,
- k is the alphabet size,
- i is the initial state,
- F is the set of final states,
- T is the set of transformations.

This notation can also be used to specify families of incomplete DFAs, i.e., $\exists_{p,s}$ such that $\delta(p, s)$ is not defined, considering the convention that, if a transformation maps a state $p \in Q$ to a state $q \notin Q$, by a symbol $s \in \Sigma$, then the transition function is not defined for state p, by symbol s.

4 Mathematical Typesetting

Since DesCo must deal with a lot of mathematical notation, due to the nature of the system, the ability to, on the one hand, typeset and display math efficiently, and on the other, allow the use of a language that is familiar to the end-user, is crucial. LaTeX is, arguably, the most popular language, amongst computer scientists, to write mathematical notation, and, as opposed to MathML, it is intended to be written and edited directly by humans. Note also that most of the mathematical formulae in DesCo are functions over integer variables (complexity measures) that can be easily coded in any programming language if any other manipulation is needed (and, actually, it can be also provided by DesCo).

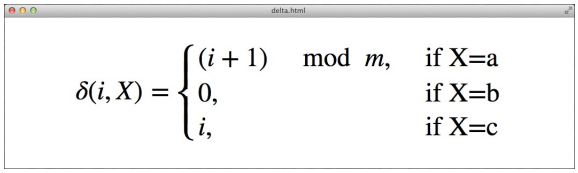

Figure 5: Web browser displaying a MathJax generated formula

Listing 1: MathJax example

```
<script src="http://cdn.mathjax.org/
mathjax/latest/MathJax.js?config=default"> </script>
$$
\delta(i,X) = \left \{
  \begin{array}{ll}
    (i+1)\mod m, & \text{if X=a} \\
    0, & \text{if X=b} \\
    i, & \text{if X=c}
  \end{array}
\right .
$$
```

All this motivates the use of MathJax [16] for the purpose of displaying math. MathJax 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 a transition function, with a branch defined for each symbol of the alphabet on a web page. The page source only needs to load MathJax and have the functions LaTeX source between $$. The code in Listing 1, generates a page, which is exhibited in Fig. 5.

5 Interrogating and Consulting

The DesCo system has a web interface for interacting with the knowledge base. This section, focuses on its data search and visualization features.

Querying the knowledge base on language classes, language families, operations, models of computation, complexity kinds or complexity measures results in a table listing the requested information. This listing is formatted using HTML/CSS and MathJax for a more aesthetically pleasing visualization. An example of such a listing is illustrated in Fig. 6.

Language families, can be further queried, for more information, such as, the code that generates its instances. Additionally, a diagram that represents a given language family, can be generated on the fly. This is exemplified in Fig. 7.

The complexity bounds can be consulted with more fine-grain detail, giving control over how the information is displayed by customizing query parameters. Firstly, one must choose the complexity kind. Then, one can choose to query all the results for a single operation. This generates a table with the complexity functions for all language classes and all complexity measures, for that specific operation, as seen in Fig. 8.

Reversal - L^R

Arity: 1

For a word x over an alphabet Σ, the *reversal* of x is denoted by x^R and it is recursively defined by:

$\epsilon^R = \epsilon$, and $(ay)^R = y^R a$ where $a \in \Sigma$ and $y \in \Sigma^*$.

By definition, if $x = a_1 \ldots a_n$, where $n \geq 1$ and $a_1, \ldots, a_n$ and are letters in Σ, the $x^R = a_n \ldots a_1$.

For a language L over an alphabet Σ, the *reversal* of L is denoted by L^R and is defined by $L^R = \{x^R \mid x \in L\}$.

Star - L^*

Arity: 1

The *Kleene star (also know as Kleene operator or Kleene closure)* of a language L can be defined as:

- $L^* = \{x_1 x_2 \ldots x_n \mid n \geq 0$ and $x_i \in L, 1 \leq i \leq n\}$

Given A and B languages, the following properties hold for the Kleene star operation.

Figure 6: Listing operations

Details for

Language Family: 7

$A = ([0, m-1], \{a, b, c\}, \delta, 0, \{m-1\})$, **and for all** $i \in [0, m-1]$,

$$\delta(i, X) = \begin{cases} (i+1) \mod m, & \text{if X=a} \\ 0, & \text{if X=b} \\ i, & \text{if X=c} \end{cases}$$

Parameter: m

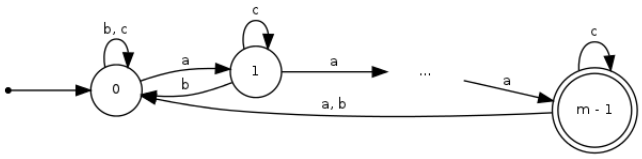

Figure 7: Language family details

Worst-Case for Union

Language Sortable	**sc** Resizable	**nsc** Resizable
Prefix-free	$mn-2$, if $m,\ n \geq 3$ $max\{m,n\}$	$m+n$
Prefix-closed	$max\{m,n\}$ mn	
Finite	$max\{m,n\}$ $mn-(m+n)$	$m+n-2$, if $m \geq 2, n \geq 2$ $max\{m,n\}$
Regular	mn , if $m > 0, n > 0$	$m+n+1$

Figure 8: Listing results for union operation

On the other hand, one can also choose a list of operations and a specific complexity measure, generating a table for each language class with all the results for the chosen operations, as presented in Fig. 9. It is also possible to list results, for all operations, for a given language class and one or more complexity measures.

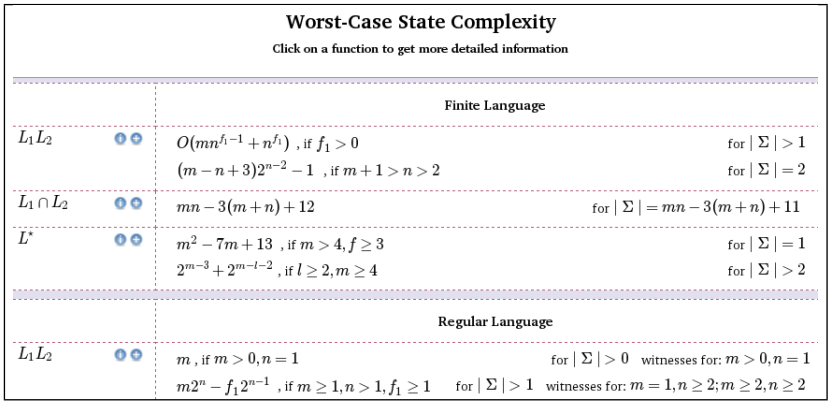

Figure 9: Listing results for several operations and language classes

All the available details, about a specific result, such as its witnesses, references and algorithms, can also be consulted, as portrayed in Fig. 10.

6 Current Integration With FAdo

The FAdo system [4] aims to provide an open source extensible software library for the symbolic manipulation of automata and other models of computation. To allow high-level programming with complex data structures, easy prototyping of algorithms, and portability, are its main features. FAdo is implemented in Python [15] and currently includes most standard operations for the manipulation of regular languages. Regular languages can be represented by regular expressions (reex) or finite automata, among other formalisms. Finite automata may be deterministic (DFA), non-deterministic (NFA) or generalized (GFA). In FAdo, these representations are implemented as Python classes. Elementary regular language operations, such as, union, intersection, concatenation, complementation, and reverse are implemented for each class. Many combined operations for DFA have specialized algorithms. Several conversions between representations are implemented: NFA to DFA via subset construction, NFA to reex using a recursive method, GFA to reex using the state elimination algorithm, with possible choice of state orderings and several heuristics, reex to NFA using the Thompson method, Glushkov method, follow, and partial derivatives. For DFA, several minimization algorithms are available: Moore, Hopcroft, Brzozowski, and some incremental algorithms. Some support is provided

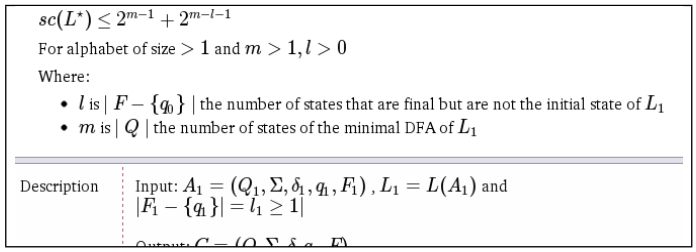

Figure 10: Details for a particular result

for computing syntactic semigroups. There are several algorithms for language equivalence. Finite languages can also be represented, by tries and AFA (acyclic finite automata).

Symbolic manipulation systems, and the FAdo system in particular, are essential for the studies of descriptional complexity, as they provide a tool for testing the complexity bounds and to find candidate witnesses. FAdo implements many of the specialized algorithms used for the upper bounds of the operational complexities of regular languages and also many of the language families used as witnesses.

The FAdo system currently aids DesCo in the generation and manipulation of language family instances. Either by using the representation using transformations, as previously defined, or by using Python code, we can generate language family instances. Once a FAdo class, representing a model of computation, has been instantiated, we can then make use of all the algorithms it has to offer to do symbolic manipulation, such as checking whether or not the first few instances of a family of DFAs are minimal, checking the equivalence of several instances of two different language families, or even verify if the given witnesses reach the upper bound for specific parameters of the language family. If these actions are too lengthy or require too many resources for the server to handle, the system can generate and export code, so the user can run it in his own machine.

7 Conclusion

DesCo is currently online (`desco.up.pt`) 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, with over three hundred results and more than sixty language families already available. Even for this subset of descriptional complexity results it is now much easier to have a general overview, analyze the complexity behavior of different operations for a language class, or compare the relative complexity of the same operation on different language classes. Work in progress includes improving the connection between the parameters of the complexity results and the ones of the language families representations (i.e, of their models of computation). Recently, the notion of *universal witness* was introduced by J. Brzozowski [6]. In DesCo we intend to allow to test if a given language family is or not a candidate to be a universal witness. To prove that a complexity bound is tight there are also other techniques besides the use of language families, for instance the *fooling-set lower bound technique* [8]. That also has to be accommodated in DesCo. Search and visualization of results may be improved in order to filter results by a language class and their subclassses, e.g. regular, context-free. Finally, the continued usage of the system by the community will suggest more improvements.

References

[1] Aaronson, S.: The Complexity Zoo (2011), `qwiki.stanford.edu/index.php/Complexity\_Zoo`

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

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

[4] Almeida, A., Almeida, M., Alves, J., Moreira, N., Reis, R.: FAdo and GUItar: tools for automata manipulation and visualization. In: Maneth, S. (ed.) 14th CIAA Proceedings. LNCS, vol. 5642, pp. 65–74. Springer, Sidney (July 2009)

[5] Brzozowski, J.A.: Quotient complexity of regular languages. J. Aut. Lang. Comb. 15(1/2), 71–89 (2010)

[6] Brzozowski, J.A.: In search of most complex regular languages. In: Moreira, N., Reis, R. (eds.) 17th CIAA Proccedings. LNCS, vol. 7381, pp. 5–24. Springer, Porto, Portugal (2012)

[7] FAdo Project: FAdo: Tools for formal languages manipulation (2011), fado.dcc.fc.up.pt

[8] Glaister, I., Shallit, J.: A lower bound technique for the size of nondeterministic finite automata. Inf. Process. Lett. 59(2), 75–77 (1996)

[9] Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. UCS 8(2), 193–234 (2002)

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

[11] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison Wesley (2006)

[12] Hromkovic, J.: Descriptional complexity of finite automata: Concepts and open problems. J. Aut. Lang. Comb. 7(4), 519–531 (2002)

[13] Kapoutsis, C.A.: minicomplexity (2012), http://minicomplexity.org

[14] Pin, J.E.: Finite semigroups and recognizable languages: an introduction. In: NATO Advanced Study Institute (1995)

[15] Python Software Foundation: Python language website. http://python.org (2012)

[16] Science, D.: MathJax (2011), www.mathjax.org

[17] Sloane, N.J.A., Plouffe, S.: The Encyclopedia of Integer Sequences. Academic Press (1995)

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

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

[20] Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. 125(2), 315–328 (1994)

[21] Zolin, E.: Navigator on description logic complexity (2011), www.cs.man.ac.uk/~ezolin/dl/