5

A Common Mathematical Framework for Asymptotic Complexity Analysis and Denotational Semantics for Recursive Programs Based on Complexity Spaces

Salvador Romaguera[1] and Oscar Valero[2]
[1]*Instituto Universitario de Matemática Pura y Aplicada, Universitat Politècnica de València*
[2]*Departamento de Ciencias Matemáticas e Informática, Universidad de las Islas Baleares Spain*

1. Introduction

In Denotational Semantics one of the aims consists of giving mathematical models of programming languages so that the meaning of a recursive algorithm can be obtained as an element of the constructed model.

Most programming languages allow to construct recursive algorithms by means of a recursive definition expressing the meaning of such a definition in terms of its own meaning. In order to analyze the correctness of such recursive definitions, D.S. Scott developed a mathematical theory of computation which is based on ideas from order theory and topology (Davey & Priestley, 1990; Scott, 1970; 1972). From the Scott theory viewpoint, the meaning of such a denotational specification is obtained as the fixed point of a nonrecursive mapping, induced by the denotational specification, which is at the same time the topological limit of successive iterations of the nonrecursive mapping acting on a distinguished element of the model. Moreover, the order of Scott's model represents some notion of information so that each iteration of the nonrecursive mapping, which models each step of the program computation, is identified with an element of the mathematical model which is greater than (or equal to) the other ones associated with the preceding iterations (preceding steps of the program computation) because each iteration gives more information about the meaning than those computed before. Hence the aforesaid meaning of the recursive denotational specification is modeled as the fixed point of the nonrecursive mapping which is obtained as the limit, with respect to the so-called Scott topology, of the increasing sequence of successive iterations. Consequently, the fixed point captures the amount of information defined by the increasing sequence, i.e. the fixed point yields the total information about the meaning provided by the elements of the increasing sequence, and it does not contain more information than can be obtained from the elements of such a sequence.

A typical and illustrative example of such recursive definitions is given by those recursive algorithms that compute the factorial of a nonnegative integer number by means of the following recursive denotational specification:

$$fact(n) = \begin{cases} 1 & \text{if } n = 1 \\ nfact(n-1) & \text{if } n > 1 \end{cases}. \quad (1)$$

Of course the above denotational specification has the drawback that the meaning of the symbol *fact* is expressed in terms of itself. Hence the symbol *fact* can not be replaced by its meaning in the denotational specification (1), since the meaning, given by the right-hand side in (1), also contains the symbol. Following the original ideas of Scott, the meaning of the specification (1), i.e. the entire factorial function, is obtained as the unique total function that is a fixed point of the nonrecursive functional ϕ_{fact} defined on the set of partial functions ordered by extension (see (Davey & Priestley, 1990) for a detailed description of the set of partial functions) by

$$\phi_{fact}f(n) = \begin{cases} 1 & \text{if } n = 1 \\ nf(n-1) & \text{if } n > 1 \text{ and } n-1 \in \text{dom}f \end{cases}',$$

where the successive iterations acting over the partial function f_1 ($\text{dom}f_1 = 1$ and $f_1(1) = 1$) hold that $\phi^n_{fact}f_1(m) = 1$ if $m = 1$ and $\phi^n_{fact}f_1(m) = m!$ for all $m \leq n$. Thus, the increasing with respect to the extension order sequence of iterations $(\phi^n_{fact}(f_1))_{n\in\mathbb{N}}$ models each step of the computation of the factorial of a nonnegative integer number by a recursive algorithm using the specification (1) and, in addition, the limit of the sequence with respect to the Scott topology is exactly the meaning of the symbol *fact* which provides the factorial of each nonnegative integer number. Furthermore, each iteration provides more information about the symbol *fact* (i.e. about the entire factorial function) than the preceding ones.

Since Scott's mathematical theory of computation was introduced, it has been wondered in the literature whether such a model can be applied to other fields of Computer Science which differ from Denotational Semantics. A positive answer to the posed question was provided by M.P. Schellekens in (Schellekens, 1995). In fact, Schellekens showed that the original Scott idea of getting, via fixed point techniques, the meaning of a denotational specification as the topological limit of "successive approximations" is helpful in Asymptotic Complexity Analysis. Concretely, Schellekens introduced a novel mathematical method to provide asymptotic upper bounds of the complexity of those algorithms whose running time of computing satisfies a recurrence equation of Divide and Conquer type in such a way that the original ideas of Scott, namely, the meaning is a fixed point and is the limit of a sequence of successive iterations of a functional acting on a distinguished element, are respected but now the fixed point technique is new. Furthermore, in Schellekens' method the topology, intrinsic to the Scott model, is induced by a "distance" tool which provides, in addition, a measure of the degree of approximation of the elements that form the model. This fact yields an advantage over the Scott model because in the latter one quantitative data approach is not available.

Motivated by the fact that Schellekens' method successfully applies the Scott ideas for Denotational Semantics to the asymptotic complexity analysis of algorithms and that, in addition, the aforesaid method improves the Scott one in the sense that it allows to provide quantitative information, not only qualitative, about the degree of approximation of the elements that form the model, the propose of this chapter is twofold.

On one hand, we will show that Schellekens' method, and thus the original ideas of Scott, is useful to obtain asymptotic upper bounds of complexity for a class of recursive algorithms whose running time of computing leads to recurrence equations different from the Divide and Conquer ones. Moreover, we improve the original Schellekens's method by introducing a new fixed point technique which allows to obtain lower asymptotic bounds for the running time of computing of the aforesaid algorithms. We will illustrate and validate the developed method applying our results to provide the asymptotic complexity (upper and lower bounds) of the running time of computing of a celebrated recursive algorithm that computes the Fibonacci sequence (see, for instance, (Cull et al., 1985)).

On the other hand, we will introduce a generalized mathematical method, in the spirit of Schellekens and based on the mathematical approach given in (Romaguera & Schellekens, 2000), which will be useful to formally describe the running time of computing of algorithms that perform a computation using recursive denotational specifications and the meaning, and thus the correctness, of such a denotational specification simultaneously. In order to validate the new results we will apply them to provide, at the same time, the program correctness and the asymptotic complexity class (upper and lower bounds) of the running time of a recursive program that computes the factorial function via the denotational specification (1).

The remainder of the chapter is organized as follows: In Section 2 we recall a few pertinent concepts from asymptotic complexity analysis of algorithms. Section 3 is devoted to introduce the method of Schellekens, and its relationship to the above exposed Scott ideas, which provides an upper bound of the asymptotic complexity for those algorithms whose running time of computing leads to a Divide and Conquer recurrence equation. The utility of the method is illustrated by means of the application given by Schellekens to obtain an asymptotic upper bound of the average running time of computing of Mergesort. In order to analyze the complexity of the recursive algorithm that computes the Fibonacci sequence via the Schellekens approach, in Section 4 we will introduce the new method based on successive approximations and fixed point techniques that allow us to describe the complexity class (asymptotic upper and lower bounds) for the running time of computing of recursive algorithms more general than the Divide and Conquer ones. Moreover, in Section 5 we give the generalized mathematical method which is useful to describe formally the running time of computing of algorithms that perform a computation using recursive denotational specifications as well as their program correctness . Finally, in Section 6 we summarize the aims achieved and the advantages of our new mathematical methodologies with respect to the Schellekens and Scott ones.

2. Fundamentals of Asymptotic Complexity Analysis

From now on, the letters $\mathbb{R}^+$ and $\mathbb{N}$ will denote the set of nonnegative real numbers and the set of positive integer numbers, respectively.

Our basic reference for complexity analysis of algorithms is (Brassard & Bratley, 1988).

In Computer Science the complexity analysis of an algorithm is based on determining mathematically the quantity of resources needed by the algorithm in order to solve the problem for which it has been designed.

A typical resource, playing a central role in complexity analysis, is the running time of computing. Since there are often many algorithms to solve the same problem, one objective of the complexity analysis is to assess which of them is faster when large inputs are considered. To this end, it is required to compare their running time of computing. This is usually done by means of the asymptotic analysis in which the running time of an algorithm is denoted by a function $T : \mathbb{N} \to (0, \infty]$ in such a way that $T(n)$ represents the time taken by the algorithm to solve the problem under consideration when the input of the algorithm is of size n. Of course the running time of an algorithm does not only depend on the input size n, but it depends also on the particular input of the size n (and the distribution of the data). Thus the running time of an algorithm is different when the algorithm processes certain instances of input data of the same size n. As a consequence, it is usually necessary to distinguish three possible behaviors when the running time of an algorithm is discussed. These are the so-called best case, the worst case and the average case. The best case and the worst case for an input of size n are defined by the minimum and the maximum running time of computing over all inputs of size n, respectively. The average case for an input of size n is defined by the expected value or average running time of computing over all inputs of size n.

In general, given an algorithm, to determine exactly the function which describes its running time of computing is an arduous task. However, in most situations is more useful to know the running time of computing of an algorithm in an "approximate" way than in an exact one. For this reason the Asymptotic Complexity Analysis focus its interest on obtaining the "approximate" running time of computing.

In order to recall the notions from Asymptotic Complexity Analysis which will be useful for our aim later on, let us assume that $f : \mathbb{N} \to (0, \infty]$ denotes the running time of computing of a certain algorithm under study. Moreover, consider that there exists a function $g : \mathbb{N} \to (0, \infty]$ such that there exist, simultaneously, $n_0 \in \mathbb{N}$ and $c \in \mathbb{R}^+$ satisfying $f(n) \leq cg(n)$ for all $n \in \mathbb{N}$ with $n \geq n_0$ ($\leq$ and $\geq$ stand for the usual orders on $\mathbb{R}^+$). Then, the function g provides an asymptotic upper bound of the running time of the studied algorithm. Hence, if we do not know the exact expression of the function f, then the function g gives an "approximate" information of the running time of the algorithm for each input size n, $f(n)$, in the sense that the algorithm takes a time to process the input data of size n bounded above by the value $g(n)$. Following the standard notation, when g is an upper asymptotic bound of f we will write $f \in \mathcal{O}(g)$.

In the analysis of the complexity of an algorithm, besides obtaining an upper asymptotic bound, it is useful to assess an asymptotic lower bound of the running time of computing. In this case the Ω-notation plays a central role. Indeed, the statement $f \in \Omega(g)$ means that there exist $n_0 \in \mathbb{N}$ and $c \in \mathbb{R}^+$ such that $cg(n) \leq f(n)$ for all $n \in \mathbb{N}$ with $n \geq n_0$. Of course, and similarly to the $\mathcal{O}$-notation case, when the time taken by the algorithm to process an input data of size n, $f(n)$, is unknown, the function g yields an "approximate" information of the running time of the algorithm in the sense that the algorithm takes a time to process the input data of size n bounded below by $g(n)$.

Of course, when the complexity of an algorithm is discussed, the best situation matches up with the case in which we can find a function $g : \mathbb{N} \to (0, \infty]$ in such a way that the running time f holds the condition $f \in \mathcal{O}(g) \cap \Omega(g)$, denoted by $f \in \Theta(g)$, since, in this case, we obtain a "tight "asymptotic bound of f and, thus, a total asymptotic information about the

time taken by the algorithm to solve the problem under consideration (or equivalently to process the input data of size n for each $n \in \mathbb{N}$). From now on, we will say that f belongs to the asymptotic complexity class of g whenever $f \in \Theta(g)$.

In the light of the above, from an asymptotic complexity analysis viewpoint, to determine the running time of an algorithm consists of obtaining its asymptotic complexity class.

3. Quasi-metric spaces and Asymptotic Complexity Analysis: the Schellekens approach

In 1995, Schellekens introduced a new mathematical framework, now known as the Complexity Space, with the aim to contribute to the topological foundation for the Asymptotic Complexity Analysis of Algorithms via the application of the original Scott ideas for Denotational Semantics, that is via the application of fixed point and successive approximations reasoning, (Schellekens, 1995). This framework is based on the notion of a quasi-metric space.

Following (Künzi, 1993), a quasi-metric on a nonempty set X is a function $d : X \times X \to \mathbb{R}^+$ such that for all $x, y, z \in X$:

(i) $d(x,y) = d(y,x) = 0 \Leftrightarrow x = y$;
(ii) $d(x,y) \leq d(x,z) + d(z,y)$.

Of course a metric on a nonempty set X is a quasi-metric d on X satisfying, in addition, the following condition for all $x, y \in X$:

(iii) $d(x,y) = d(y,x)$.

A quasi-metric space is a pair (X,d) such that X is a nonempty set and d is a quasi-metric on X.

Each quasi-metric d on X generates a T_0-topology $\mathcal{T}(d)$ on X which has as a base the family of open d-balls $\{B_d(x,\varepsilon) : x \in X, \varepsilon > 0\}$, where $B_d(x,\varepsilon) = \{y \in X : d(x,y) < \varepsilon\}$ for all $x \in X$ and $\varepsilon > 0$.

Given a quasi-metric d on X, the function d^s defined on $X \times X$ by

$$d^s(x,y) = \max\left(d(x,y), d(y,x)\right),$$

is a metric on X.

A quasi-metric space (X,d) is called bicomplete whenever the metric space (X,d^s) is complete.

A well-known example of a bicomplete quasi-metric space is the pair $((0,\infty], u_{-1})$, where

$$u_{-1}(x,y) = \max\left(\frac{1}{y} - \frac{1}{x}, 0\right),$$

for all $x, y \in (0,\infty]$. Obviously we adopt the convention that $\frac{1}{\infty} = 0$. The quasi-metric space $((0,\infty], u_{-1})$ plays a central role in the Schellekens framework. Indeed, let us recall that the

complexity space is the pair $(\mathcal{C}, d_{\mathcal{C}})$, where

$$\mathcal{C} = \{f : \mathbb{N} \to (0, \infty] : \sum_{n=1}^{\infty} 2^{-n} \frac{1}{f(n)} < \infty\},$$

and $d_{\mathcal{C}}$ is the quasi-metric (so-called complexity distance) on $\mathcal{C}$ defined by

$$d_{\mathcal{C}}(f,g) = \sum_{n=1}^{\infty} 2^{-n} u_{-1}(f(n), g(n)) = \sum_{n=1}^{\infty} 2^{-n} \max\left(\frac{1}{g(n)} - \frac{1}{f(n)}, 0\right).$$

According to (Schellekens, 1995), since every reasonable algorithm, from a computability viewpoint, must hold the "convergence condition" $\sum_{n=1}^{\infty} 2^{-n} \frac{1}{f(n)} < \infty$, it is possible to associate each algorithm with a function of $\mathcal{C}$ in such a way that such a function represents, as a function of the size of the input data, its running time of computing. Motivated by this fact, the elements of $\mathcal{C}$ are called complexity functions. Moreover, given two functions $f, g \in \mathcal{C}$, the numerical value $d_{\mathcal{C}}(f,g)$ (the complexity distance from f to g) can be interpreted as the relative progress made in lowering the complexity by replacing any algorithm P with complexity function f by any algorithm Q with complexity function g. Therefore, if $f \neq g$, the condition $d_{\mathcal{C}}(f,g) = 0$ can be read as f is "at least as efficient" as g on all inputs (note that $d_{\mathcal{C}}(f,g) = 0 \Leftrightarrow f(n) \leq g(n)$ for all $n \in \mathbb{N}$). In fact, the condition $d_{\mathcal{C}}(f,g) = 0$ provides that $f \in \mathcal{O}(g)$.

Notice that the asymmetry of the complexity distance $d_{\mathcal{C}}$ plays a central role in order to provide information about the increase of complexity whenever an algorithm is replaced by another one. A metric will be able to yield information on the increase but it, however, will not reveal which algorithm is more efficient.

The utility of the complexity space in complexity analysis of algorithms was also illustrated by Schellekens in (Schellekens, 1995). In particular, he introduced a mathematical method to provide asymptotic upper bounds of the running time of computing for Divide and Conquer algorithms. To this end, Schellekens used the below fixed point theorem which is a quasi-metric version of the celebrate Banach's fixed point theorem.

Theorem 1. *Let f be a contractive mapping from a bicomplete quasi-metric space (X,d) into itself with contractive constant s. Then f has a unique fixed point x_0 and, in addition,* $\lim_{n\to\infty} f^n(x) = x_0$ *for all $x \in X$.*

Let us remark that a mapping f from a quasi-metric space (X,d) into itself it is called contractive provided that there exists $s \in [0,1)$ such that for all $x, y \in X$

$$d(f(x), f(y)) \leq s d(x,y).$$

In this case, s is called a contractive constant of f.

Next we recall the aforenamed method with the aim of motivating our subsequent work, given in Sections 4 and 5.

A Divide and Conquer algorithm solves a problem of size n ($n \in \mathbb{N}$) splitting it into a subproblems of size $\frac{n}{b}$, for some constants a, b with $a, b \in \mathbb{N}$ and $a, b > 1$, and solving them separately by the same algorithm. After obtaining the solution of the subproblems, the

algorithm combines all subproblem solutions to give a global solution to the original problem. The recursive structure of a Divide and Conquer algorithm leads to a recurrence equation for the running time of computing. In many cases the running time of a Divide and Conquer algorithm is the solution to a Divide and Conquer recurrence equation, that is a recurrence equation of the form

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ aT(\frac{n}{b}) + h(n) & \text{if } n \in \mathbb{N}_b \end{cases}, \tag{2}$$

where $\mathbb{N}_b = \{b^k : k \in \mathbb{N}\}$, $c > 0$ denotes the complexity on the base case (i.e. the problem size is small enough and the solution takes constant time), and $h(n)$ represents the time taken by the algorithm in order to divide the original problem into a subproblems and to combine all subproblems solutions into a unique one ($h \in \mathcal{C}$ with $h(n) < \infty$ for all $n \in \mathbb{N}$).

Notice that for Divide and Conquer algorithms, it is typically sufficient to obtain the complexity on inputs of size n with n ranges over the set $\mathbb{N}_b$ (see (Brassard & Bratley, 1988) for a fuller description).

Mergesort (in all behaviors) and Quicksort (in the best case behavior) are typical and well-known examples of Divide and Conquer algorithms whose running time of computing satisfies the recurrence equation (2) (we refer the reader to (Brassard & Bratley, 1988) and (Cull et al., 1985) for a detailed discussion about the both aforasaid algorithms).

In order to provide the asymptotic behavior of the running time of computing of a Divide and Conquer algorithm satisfying the recurrence equation (2), it is necessary to show that such a recurrence equation has a unique solution and, later, to obtain the asymptotic complexity class of such a solution. The method introduced by Schellekens allows us to show that the equation (2) has a unique solution, and provides an upper asymptotic complexity bound of the solution in the following way:

Denote by $\mathcal{C}_{b,c}$ the subset of $\mathcal{C}$ given by

$$\mathcal{C}_{b,c} = \{f \in \mathcal{C} : f(1) = c \text{ and } f(n) = \infty \text{ for all } n \in \mathbb{N}\backslash\mathbb{N}_b \text{ with } n > 1\}.$$

Since the quasi-metric space $(\mathcal{C}, d_{\mathcal{C}})$ is bicomplete (see Theorem 3 and Remark in page 317 of (Romaguera & Schellekens, 1999)) and the set $\mathcal{C}_{b,c}$ is closed in $(\mathcal{C}, d^s_{\mathcal{C}})$, we have that the quasi-metric space $(\mathcal{C}_{b,c}, d_{\mathcal{C}}|_{\mathcal{C}_{b,c}})$ is bicomplete.

Next we associate a functional $\Phi_T : \mathcal{C}_{b,c} \longrightarrow \mathcal{C}_{b,c}$ with the Divide and Conquer recurrence equation (2) as follows:

$$\Phi_T(f)(n) = \begin{cases} c & \text{if } n = 1 \\ \infty & \text{if } n \in \mathbb{N}\backslash\mathbb{N}_b \text{ and } n > 1 \\ af(\frac{n}{b}) + h(n) & \text{otherwise} \end{cases}. \tag{3}$$

Of course a complexity function in $\mathcal{C}_{b,c}$ is a solution to the recurrence equation (2) if and only if it is a fixed point of the functional Φ_T. Under these conditions, Schellekens proved (Schellekens, 1995) that

$$d_{\mathcal{C}}|_{\mathcal{C}_{b,c}}(\Phi_T(f), \Phi_T(g)) \leq \frac{1}{a} d_{\mathcal{C}}|_{\mathcal{C}_{b,c}}(f, g), \tag{4}$$

for all $f, g \in \mathcal{C}_{b,c}$. So, by Theorem 1, the functional $\Phi_T : \mathcal{C}_{b,c} \longrightarrow \mathcal{C}_{b,c}$ has a unique fixed point and, thus, the recurrence equation (2) has a unique solution.

In order to obtain the asymptotic upper bound of the solution to the recurrence equation (2), Schellekens introduced a special class of functionals known as improvers.

Let $C \subseteq \mathcal{C}$. A functional $\Phi : C \longrightarrow C$ is called an improver with respect to a function $f \in C$ provided that $\Phi^n(f) \leq \Phi^{n-1}(f)$ for all $n \in \mathbb{N}$. Of course $\Phi^0(f) = f$.

Observe that an improver is a functional which corresponds to a transformation on algorithms in such a way that the iterative applications of the transformation yield, from a complexity point of view, an improved algorithm at each step of the iteration.

Taking into account the exposed facts, Schellekens stated the following result in (Schellekens, 1995).

Theorem 2. *A Divide and Conquer recurrence of the form (2) has a unique solution f_T in $\mathcal{C}_{b,c}$ suh that $\lim_{n\to\infty} \Phi_T^n(g) = f_T$ for all $g \in \mathcal{C}_{b,c}$. Moreover, if the functional Φ_T associated with (2) is an improver with respect to some function $g \in \mathcal{C}_{b,c}$, then the solution to the recurrence equation satisfies that $f_T \in \mathcal{O}(g)$.*

Of course the preceding theorem states a few relationships between the Schellekens framework and the Scott one (see Section 1). Concretely, the solution to a recurrence equation of type (2) is the fixed point $f_T \in \mathcal{C}_{b,c}$ of the nonrecursive functional Φ_T, which can be seen as the topological limit of the sequence of the successive iterations $(\Phi_T^n(g))_{n\in\mathbb{N}}$ where Φ_T is an improver with respect to $g \in \mathcal{C}_{b,c}$. Note that in this case the functional Φ_T plays the role of the nonrecursive functional ϕ_{fact} introduced in Section 1 and that the role of meaning of the factorial recursive denotational specification, i.e. the factorial function, is now played by the running time of computing f_T. Moreover, the facts that functional Φ_T is an improver and $\lim_{n\to\infty} \Phi_T^n(g) = f_T$ in $(\mathcal{C}_{b,c}, d^s|_{\mathcal{C}_{b,c}})$ yield that each iteration gives more information about the solution to the recurrence equation (the running time of computing) than the preceding ones and, in addition, one has that the information about the running time of computing of the algorithm under study is exactly that which can be obtained from the elements of such a sequence. Hence the role of the distinguished element f_1 and the successive approximations sequence $(\phi^n(f_1))_{n\in\mathbb{N}}$ of the Scott framework, is now played by the sequence of improved running time versions of the Divide and Conquer algorithm under study, $(\Phi_T^n(g))_{n\in\mathbb{N}}$, and the complexity function $g \in \mathcal{C}_{b,c}$ with respect to which the nonrecursive functional Φ_T is an improver. These analogies between the Schellekens and Scott techniques give relevance to the former one with respect to the standard and classical techniques to analyze the complexity of algorithms. Simultaneously, the framework based on the complexity space presents an advantage with respect to the Scott one. Indeed, the framework of Schellekens counts on a topology induced by a quasi-metric (the complexity distance) which allows to measure the information about the fixed point of the nonrecursive functional contained in each element of the sequence of successive approximations and the framework of Scott has not available, in general, quantitative data approach.

In order to validate Theorem 2 and to illustrate its usefulness, Schellekens obtained an upper asymptotic bound of the running time of Mergesort (in the average case behavior). In this particular case, the Mergesort running time of computing satisfies the following particular

case of recurrence equation (2):

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(\frac{n}{2}) + \frac{n}{2} & \text{if } n \in \mathbb{N}_2 \end{cases} . \tag{5}$$

It is clear that Theorem 2 shows that the recurrence equation (5) has a unique solution f_T^M in $\mathcal{C}_{2,c}$. In addition, it is not hard to check that the functional Φ_T induced by the recurrence equation (5), and given by (3), is an improver with respect to a complexity function $g_k \in \mathcal{C}_{2,c}$ (with $k > 0$, $g_k(1) = c$ and $g_k(n) = kn \log_2 n$ for all $n \in \mathbb{N}_2$) if and only if $k \geq \frac{1}{2}$. Note that under the assumption that the functional Φ in (3) is monotone, this is the case for the functional Φ_T induced by (5), to show that Φ_T is an improver with respect to $f \in C$ is equivalent to verify that $\Phi_T(f) \leq f$.

Therefore, by Theorem 2, we can conclude that $f_T^M \in \mathcal{O}(g_{\frac{1}{2}})$, i.e. Theorem 2 provides a new formal proof, inspired by the original Scott fixed point approach, of the well-known fact that the running time of computing f_T^M of Mergesort (in the average case behavior) belongs to $\mathcal{O}(n \log_2 n)$, i.e. that the complexity function $g_{\frac{1}{2}}$, or equivalently $\mathcal{O}(n \log_2 n)$, gives an asymptotic upper bound of f_T^M.

Furthermore, it must be stressed that in (Schellekens, 1995) it was pointed out that an asymptotic lower bound of the running time of Mergesort (in the average case behavior) belongs to $\Omega(n \log_2 n)$. However, to show this standard arguments, which are not based on the use of fixed point techniques, were followed. So Schellekens proved that Mergesort running time (in the average case behavior) belongs to the complexity class $\Theta(n \log_2 n)$, but the unique, strictly speaking, novel proof of the last fact was given when the asymptotic upper bound was obtained.

4. An extension of Schellekens' approach: the general case of recursive algorithms

Although the most natural is to think that the running time of computing of Divide and Conquer algorithms is always the solution to a Divide and Conquer recurrence equations of type (2), there are well-known examples, like Quicksort (worst case behaviour), of Divide and Conquer algorithms whose complexity analysis does not lead with a recurrence equation of the aforesaid type (Cull et al., 1985). In particular, the running time of computing (worst case behavior) for the aforenamed algorithm is the solution to the recurrence equation given by

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ T(n-1) + jn & \text{if } n \geq 2 \end{cases}, \tag{6}$$

with $j > 0$ and where c is the time taken by the algorithm in the base case. Observe that in this case it is not necessary to restrict the input size of the data to a set $\mathbb{N}_b$ as defined in the preceding section.

Clearly, the recurrence equation (6) can not be retrieved as a particular case of the Divide and Conquer family of recurrence equations (2). However, the main and strong relationship between Mergesort and Quicksort is given by the fact that both are recursive algorithms. Obviously, the class of recursive algorithms is wider than the Divide and Conquer one. An

illustrative example, which does not belong to the Divide and Conquer family, is provided by the recursive algorithm, which we will call Hanoi, that solves the Towers of Hanoi puzzle (Cull et al., 1985; Cull & Ecklund, 1985). In this case, under the uniform cost criterion assumption, the running time of computing is the solution to a recurrence equation given by

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n-1) + d & \text{if } n \geq 2 \end{cases}, \tag{7}$$

where $c, d > 0$ and where c represents the time taken by the algorithm to solve the base case. Of course, to distinguish three possible running time behaviors for Hanoi is meaningless, since the input data distribution is always the same for each size n.

In (Romaguera et al., 2011), the fact that the class of recursive algorithms is wider than the Divide and Conquer inspired to wonder whether one can obtain a family of recurrence equations in such a way that the complexity analysis of those algorithms whose running time of computing is a solution either to recurrence equations associated with Quicksort (in the worst case behavior) and Hanoi or to a Divide and Conquer one can be carried out from it and, in addition, whether such a complexity analysis can be done via an extension of the fixed point technique of Schellekens.

A positive answer to the preceding questions was also given in (Romaguera et al., 2011). Concretely, it was shown two things.

On one hand, it was pointed out that the recurrence equations that yield the running time of computing of the above aforesaid algorithms can be considered as particular cases of the following general one:

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ aT(n-1) + h(n) & \text{if } n \geq 2 \end{cases}, \tag{8}$$

where $c > 0$, $a \geq 1$ and $h \in \mathcal{C}$ such that $h(n) < \infty$ for all $n \in \mathbb{N}$.

Of course, the discussion of the complexity of the Divide and Conquer algorithms introduced in Section 3 can be carried out from the family of recurrence equations of type (8). This is possible because the running time of computing of the aforementioned algorithms leads to recurrence equations can be seen as a particular case of our last general family of recurrence equations. Indeed, a Divide and Conquer recurrence equation

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ aT(\frac{n}{b}) + h(n) & \text{if } n \in \mathbb{N}_b \end{cases}, \tag{9}$$

can be transformed into the following one

$$S(m) = \begin{cases} c & \text{if } m = 1 \\ aS(m-1) + r(m) & \text{if } m > 1 \end{cases}, \tag{10}$$

where $S(m) = T(b^{m-1})$ and $r(m) = h(b^{m-1})$ for all $m \in \mathbb{N}$. (Recall that $\mathbb{N}_b = \{b^k : k \in \mathbb{N}\}$ with $b \in \mathbb{N}$ and $b > 1$).

Observe that the analysis of the recurrence equation family (10) allows immediately to study the Divide and Conquer recurrence equations.

On the other hand, the Schellekens fixed point technique was extended and applied to discuss the complexity class (asymptotic upper and lower bounds) of those algorithms whose running time of computing is the solution to a recurrence equation of type (8). Of course, it was introduced an improvement over the original Schellekens method in order to obtain asymptotic lower bounds of the running time of computing of algorithms. This was done by means of a new kind of functionals defined on the complexity space and called worseners (see Subsection 4.2 for the definition). Furthermore, the applicability of the new results to Asymptotic Complexity Analysis was illustrated by discussing the complexity class of the running time of computing, among others, of Quicksort (in the worst case behavior) and Hanoi.

Nevertheless, a recursive algorithm whose running time does not hold a recurrence equation of type (8) can be found, for instance, in (Cull et al., 1985). The aforementioned algorithm, that we will call Fibonacci, computes the value of the Fibonacci sequence at any given index n with $n \in \mathbb{N}$. The running time of computing of Fibonacci matches up with the solution to a recurrence equation given as follows:

$$T(n) = \begin{cases} 2c & \text{if } n = 1 \\ 3c & \text{if } n = 2 \\ T(n-1) + T(n-2) + 4c & \text{if } n > 2 \end{cases}, \tag{11}$$

where c represents the time taken by the computer to perform a basic operation. Again, similarly to Hanoi, to distinguish three possible running time behaviors is meaningless.

It is clear that the recurrence equation (11) can not be retrieved as a particular case from the family of recurrence equations (8). Consequently, the latter family of recurrence equations is not be able to model the complexity of Fibonacci. However, the running time of computing of all above recursive algorithms, including Fibonacci, can be modeled as the solution to a recurrence equation of the below general family:

$$T(n) = \begin{cases} c_n & \text{if } 1 \leq n \leq k \\ \sum_{i=1}^{k} a_i T(n-i) + h(n) & \text{if } n > k \end{cases}, \tag{12}$$

where $h \in \mathcal{C}$ such that $h(n) < \infty$ for all $n \in \mathbb{N}$, $k \in \mathbb{N}$, $c_i > 0$ and $a_i \geq 1$ for all $1 \leq i \leq k$.

Inspired by the previous exposed fact our purpose in this section is to go more deeply into the Schellekens fixed point technique for the complexity analysis of algorithms and to demonstrate that such techniques can be successfully used to obtain the complexity class of those recursive algorithms whose running time is a solution to a recurrence equation of type (12). In particular we prove, by means of a new fixed point theorem, the existence and uniqueness of the solution to a recurrence equation of type (12). Moreover, we introduce, based on the fixed point theorem, a technique in the spirit of Schellekens and Scott to get the the complexity class (an asymptotic upper and lower bound) of such a solution. Concretely, our technique to obtain asymptotic upper bounds is based on, following the Schellekens original ideas, the use of improver functionals induced by the recurrence equation. Nevertheless, following (Romaguera et al., 2011), we provide asymptotic lower bounds of the solution via worseners functionals which capture, similarly to the improvers one but in a dual way, the original "successive approximations" spirit of Scott's approach.

Furthermore, we prove that in order to provide the complexity class of an algorithm whose running time satisfies a recurrence equation of type (12) is enough to search among all complexity functions for which the functional associated to the recurrence equation is an improver and a worsener simultaneously. Finally, with the aim, on one hand, to validate our new results and, on the other hand, to show the potential applicability of the developed theory to Asymptotic Complexity Analysis, we end the section discussing the complexity class of the running time of Fibonacci.

4.1 The fixed point: existence and uniqueness of solution

Fix $k \in \mathbb{N}$. Consider the subset $\mathcal{C}_{c,k}$ of $\mathcal{C}$ given by

$$\mathcal{C}_{c,k} = \{f \in \mathcal{C} : f(n) = c_n \text{ for all } 1 \leq n \leq k\}.$$

Define the functional $\Psi_T : \mathcal{C}_{c,k} \longrightarrow \mathcal{C}_{c,k}$ by

$$\Psi_T(f)(n) = \begin{cases} c_n & \text{if } 1 \leq n \leq k \\ \sum_{i=1}^{k} a_i f(n-i) + h(n) & \text{if } n > k \end{cases}, \tag{13}$$

for all $f \in \mathcal{C}_{c,k}$.

It is clear that a complexity function in $\mathcal{C}_{c,k}$ is a solution to a recurrence equation of type (12) if and only if it is a fixed point of the functional Ψ_T.

The next result is useful to supply the bicompleteness of the quasi-metric space $(\mathcal{C}_{c,k}, d_{\mathcal{C}}|_{\mathcal{C}_{c,k}})$.

Proposition 3. *The subset $\mathcal{C}_{c,k}$ is closed in $(\mathcal{C}, d^s_{\mathcal{C}})$.*

Proof. Let $g \in \overline{\mathcal{C}_{c,k}}^{d^s_{\mathcal{C}}}$ and $(f_i)_{i\in\mathbb{N}} \subset \mathcal{C}_{c,k}$ with $\lim_{i\to\infty} d^s_{\mathcal{C}}(g, f_i) = 0$. First of all we prove that $g \in \mathcal{C}$. Indeed, given $\varepsilon > 0$, there exist $i_0, n_0 \in \mathbb{N}$ such that $d^s_{\mathcal{C}}(g, f_i) < \varepsilon$ whenever $i \geq i_0$ and $\sum_{n=n_0+1}^{\infty} 2^{-n} \frac{1}{f_{i_0}(n)} < \varepsilon$. Whence

$$\begin{aligned} \sum_{n=n_0+1}^{\infty} 2^{-n} \frac{1}{g(n)} &= \sum_{n=n_0+1}^{\infty} 2^{-n} \left(\frac{1}{g(n)} - \frac{1}{f_{i_0}(n)} + \frac{1}{f_{i_0}(n)} \right) \\ &\leq \sum_{n=n_0+1}^{\infty} 2^{-n} \left| \frac{1}{g(n)} - \frac{1}{f_{i_0}(n)} \right| + \sum_{n=n_0+1}^{\infty} 2^{-n} \frac{1}{f_{i_0}(n)} \\ &\leq 2 d^s_{\mathcal{C}}(g, f_{i_0}) + \sum_{n=n_0+1}^{\infty} 2^{-n} \frac{1}{f_{i_0}(n)} \\ &< 3\varepsilon. \end{aligned}$$

Now suppose for the purpose of contradiction that $g \notin \mathcal{C}_{c,k}$. Then there exists $n_1 \in \mathbb{N}$ with $1 \leq n_1 \leq k$ such that $g(n_1) \neq c_{n_1}$. Put $\varepsilon = 2^{-n_1} |\frac{1}{g(n_1)} - \frac{1}{c_{n_1}}|$. Then there exists $i_0 \in \mathbb{N}$ such that $d^s_{\mathcal{C}}(g, f_i) < \frac{\varepsilon}{2}$ whenever $i \geq i_0$. Thus

$$\sum_{n=1}^{\infty} 2^{-n} \left| \frac{1}{g(n)} - \frac{1}{f_i(n)} \right| < \varepsilon,$$

whenever $i \geq i_0$. As a result we have that

$$\varepsilon = 2^{-n_1}|\frac{1}{g(n_1)} - \frac{1}{c_{n_1}}| \leq \sum_{n=1}^{\infty} 2^{-n} \left| \frac{1}{g(n)} - \frac{1}{f_{i_0}(n)} \right| < \varepsilon,$$

which is a contradiction. So $g(n) = c_n$ for all $1 \leq n \leq k$. Therefore we have shown that $\overline{\mathcal{C}_{c,k}}^{d^s_{\mathcal{C}}} = \mathcal{C}_{c,k}$ and, thus, that $\mathcal{C}_{c,k}$ is closed in $(\mathcal{C}, d^s_{\mathcal{C}})$. ■

Since the metric space $(\mathcal{C}, d^s_{\mathcal{C}})$ is complete and, by Proposition 3, the subset $\mathcal{C}_{c,k}$ is closed in $(\mathcal{C}, d^s_{\mathcal{C}})$ we immediately obtain the following consequence.

Corollary 4. *The quasi-metric space $(\mathcal{C}_{c,k}, d_{\mathcal{C}}|_{\mathcal{C}_{c,k}})$ is bicomplete.*

The next result provides the existence and uniqueness of solution to a recurrence equation of type (12).

Theorem 5. *The functional Ψ_T is contractive from $(\mathcal{C}_{c,k}, d_{\mathcal{C}}|_{\mathcal{C}_{c,k}})$ into itself with contractive constant*

$$(\max_{1\leq i\leq k} \frac{1}{a_i}) \left(\frac{2^k - 1}{2^k} \right).$$

Proof. Let $f, g \in \mathcal{C}_{c,k}$. Then we prove that

$$d_{\mathcal{C}}|_{\mathcal{C}_{c,k}}(\Psi_T(f), \Psi_T(g)) \leq s d_{\mathcal{C}}|_{\mathcal{C}_{c,k}}(f, g),$$

where

$$s = (\max_{1\leq i\leq k} \frac{1}{a_i}) \left(\frac{2^k - 1}{2^k} \right).$$

Indeed, we have that

$$\begin{aligned}
d_{\mathcal{C}}|_{\mathcal{C}_{c,k}}(\Psi_T(f), \Psi_T(g)) &= \sum_{n=1}^{\infty} 2^{-n} \max\left(\frac{1}{\Psi_T(g)(n)} - \frac{1}{\Psi_T(f)(n)}, 0 \right) \\
&= \sum_{n=k+2}^{\infty} 2^{-n} \max\left(\frac{\sum_{i=1}^{k} a_i \left(f(n-i) - g(n-i) \right)}{r(n)}, 0 \right) \\
&\leq \sum_{n=k+2}^{\infty} 2^{-n} \max\left(\frac{\sum_{i=1}^{k} a_i \left(f(n-i) - g(n-i) \right)}{\sum_{i=1}^{k} a_i^2 \left(g(n-i) f(n-i) \right)}, 0 \right) \\
&\leq (\max_{1\leq i\leq k} \frac{1}{a_i}) \sum_{n=k+2}^{\infty} 2^{-n} \max\left(\sum_{i=1}^{k} \frac{1}{g(n-i)} - \frac{1}{f(n-i)}, 0 \right) \\
&\leq (\max_{1\leq i\leq k} \frac{1}{a_i}) \sum_{n=k+1}^{\infty} 2^{-n} \left(\frac{2^k - 1}{2^k} \right) \max\left(\frac{1}{g(n)} - \frac{1}{f(n)}, 0 \right) \\
&= (\max_{1\leq i\leq k} \frac{1}{a_i}) \left(\frac{2^k - 1}{2^k} \right) d_{\mathcal{C}}|_{\mathcal{C}_{c,k}}(f, g),
\end{aligned}$$

where

$$r(n) = (h(n))^2 + h(n)\sum_{i=1}^{k} a_i\,(f(n-i)+g(n-i)) + \left(\sum_{i=1}^{k} a_i f(n-i)\right)\cdot\left(\sum_{i=1}^{k} a_i g(n-i)\right),$$

for all $n > k$.

Now the existence and uniqueness of the fixed point $f_T \in \mathcal{C}_{c,k}$ of Ψ_T follow from Corollary 4 and Theorem 1, because $(\max_{1\leq i\leq k} \frac{1}{a_i})\left(\frac{2^k-1}{2^k}\right) < 1$. ■

Since $f_T \in \mathcal{C}_{c,k}$ is the solution to a the recurrence equation of type (12) if and only if f_T is a fixed point of Ψ_T, Theorem 5 yields that a recurrence of the form (12) has a unique solution f_T in $\mathcal{C}_{c,k}$. Moreover, note that, by Theorem 1, we have that $\lim_{n\to\infty} \Psi_T^n(g) = f_T$ in $(\mathcal{C}_{c,k}, d^s|_{\mathcal{C}_{c,k}})$ for all $g \in \mathcal{C}_{c,k}$.

4.2 Bounding the solution: the complexity class of running time of computing

In order to describe the complexity of those recursive algorithms whose running time of computing satisfies a recurrence equation of type (12) we need recall the below auxiliary result which was announced without proof in (Romaguera et al., 2011) with the aim of extending Schellekens fixed point technique and, thus, analyze the complexity class of those algorithms whose running time of computing holds a recurrence equation of type (8) .

Lemma 6. *Let C be a subset of $\mathcal{C}$ such that the quasi-metric space $(C, d_{\mathcal{C}}|_C)$ is bicomplete and suppose that $\Psi : C \longrightarrow C$ is a contractive functional with fixed point $f \in C$ and contractive constant s. Then the following statements hold:*

1) If there exists $g \in C$ with $d_{\mathcal{C}}|_C(\Psi(g), g) = 0$, then $d_{\mathcal{C}}|_C(f, g) = 0$.
2) If there exists $g \in C$ with $d_{\mathcal{C}}|_C(g, \Psi(g)) = 0$, then $d_{\mathcal{C}}|_C(g, f) = 0$.

Proof. 1) Assume that there exists $g \in C$ such that $d_{\mathcal{C}}|_C(\Psi(g), g) = 0$. Suppose for the purpose of contradiction that $d_{\mathcal{C}}|_C(f, g) > 0$. Then we have that

$$\begin{aligned} d_{\mathcal{C}}|_C(f,g) &\leq d_{\mathcal{C}}|_C(f,\Psi(g)) + d_{\mathcal{C}}|_C(\Psi(g),g) = d_{\mathcal{C}}|_C(f,\Psi(g)) \\ &\leq d_{\mathcal{C}}|_C(f,\Psi(f)) + d_{\mathcal{C}}|_C(\Psi(f),\Psi(g)) \\ &= d_{\mathcal{C}}|_C(\Psi(f),\Psi(g)) \leq s d_{\mathcal{C}}|_C(f,g). \end{aligned}$$

From the preceding inequality we deduce that $1 \leq s$, which contradicts the fact that s is a contractive constant. So $d_{\mathcal{C}}|_C(f, g) = 0$.

2) The thesis of 2) can be proved applying similar arguments to those given in the proof of 1). ■

Observe that if a complexity function f represents the running time of computing of an algorithm under study, the fact that there exists a complexity function g satisfying the condition $d_{\mathcal{C}}|_C(\Psi(g), g) = 0$ ($d_{\mathcal{C}}|_C(g, \Psi(g)) = 0$) in the preceding lemma provides an asymptotic upper (lower) bound of the aforesaid running time, since $d_{\mathcal{C}}|_C(f, g) = 0$ ($d_{\mathcal{C}}|_C(g, f) = 0$) implies that $f \in \mathcal{O}(g)$ ($f \in \Omega(g)$).

From statements in Lemma 6 we infer, such as it was pointed out in (Romaguera et al., 2011), that to provide an asymptotic upper bound of the running time of computing of an algorithm whose running time matches up with the fixed point of a contractive mapping $\Psi : C \longrightarrow C$ ($C \subseteq \mathcal{C}$) associated to a recurrence equation is enough to check if such a mapping satisfies the condition $\Psi(g) \leq g$ for any complexity function even if Ψ is not monotone (see Section 3). This fact presents an improvement of our new method over the Schellekens one. So in the remainder of this paper, and according to (Romaguera et al., 2011), given $C \subseteq \mathcal{C}$ and a contraction $\Psi : C \longrightarrow C$ we will say that Ψ is a cont-improver with respect to a complexity function $g \in C$ provided that $\Psi(g) \leq g$. Note that the contractivity of the functional implies that a cont-improver is an improver. Consequently, a cont-improver is a transformation on algorithms in such a way that the application of the transformation yields, from a complexity point of view, an improved algorithm.

Motivated by statement 2) in Lemma 6, it was introduced a new kind of functionals called worseners. For our subsequent purpose, let us recall that, given $C \subseteq \mathcal{C}$, a contraction $\Psi : C \longrightarrow C$ is called a *worsener* with respect to a function $f \in C$ provided that $f \leq \Psi(f)$. An easy verification shows that if a functional Ψ is a worsener with respect to f, then $\Psi^{n-1}(f) \leq \Psi^n(f)$ for all $n \in \mathbb{N}$. Hence the computational meaning of a worsener functional is dual to the meaning of a cont-improver. Indeed, a worsener is a functional which corresponds to a transformation on algorithms in such a way that the application of the transformation yields, from a complexity point of view, a worsened algorithm.

The next result provides the method to provide the asymptotic upper and lower bounds of the running time of computing of those algorithms whose running time of computing is the solution to a recurrence equation of type (12).

Theorem 7. *Let $f_T \in \mathcal{C}_{c,k}$ be the (unique) solution to a recurrence equation of type (12). Then the following facts hold:*

1) *If the functional Ψ_T associated to (12), and given by (13), is a cont-improver with respect to some function $g \in \mathcal{C}_{c,k}$, then $f_T \in \mathcal{O}(g)$.*
2) *If the functional Ψ_T associated to (12), and given by (13), is a worsener with respect to some function $g \in \mathcal{C}_{c,k}$, then $f_T \in \Omega(g)$.*

Proof. Assume that Ψ_T is a cont-improver with respect to $g \in \mathcal{C}_{c,k}$. Then we have $\Psi_T(g) \leq g$. Hence we obtain that $d_{\mathcal{C}}|_{\mathcal{C}_{c,k}}(\Psi_T(g), g) = 0$. It immediately follows, by statement 1) in Lemma 6, that $d_{\mathcal{C}}|_{\mathcal{C}_{c,k}}(f_T, g) = 0$ and, thus, $f_T \in \mathcal{O}(g)$. So we have proved 1).

To prove 2) suppose that Ψ_T is a worsener with respect to $g \in \mathcal{C}_{c,k}$. Then $g \leq \Psi_T(g)$. Whence we deduce that $d_{\mathcal{C}}|_{\mathcal{C}_{c,k}}(g, \Psi_T(g)) = 0$. Thus statement 2) in Lemma 6 gives that $d_{\mathcal{C}}|_{\mathcal{C}_{c,k}}(g, f_T) = 0$, and we conclude that $f_T \in \Omega(g)$. This finishes the proof. ■

In the light of Theorem 7, we have that the solution to a recurrence equation of type (12) satisfies that $f_T \in \mathcal{O}(g) \cap \Omega(h)$ whenever Ψ_T is a cont-improver and a worsener with respect to $g \in \mathcal{C}_{c,k}$ and $h \in \mathcal{C}_{c,k}$, respectively. Consequently we obtain the complexity class of algorithms whose running time of computing satisfies a recurrence equation of type (12) when there exist $l \in \mathcal{C}_{c,k}$, $r, t > 0$ and $n_0 \in \mathbb{N}$ such that $g(n) = rl(n)$ and $h = tl(n)$ for all $n \geq n_0$ and, besides, Ψ_T is a cont-improver and a worsener with respect to g and h respectively, because, in such a case, $f_T \in \Theta(l)$.

4.3 An application to Asymptotic Complexity Analysis: Fibonacci case

In this subsection we validate the developed theory and illustrate the potential applicability of the obtained results by means of providing the asymptotic complexity class of the recursive algorithm that computes the Fibonacci sequence, an algorithm whose running time of computing can be analyzed neither via the technique exposed in (Schellekens, 1995) nor via its extended version given in (Romaguera et al., 2011).

As we have exposed above, the running time of computing of Fibonacci is the solution to the below recurrence equation:

$$T(n) = \begin{cases} 2c & \text{if } n = 1 \\ 3c & \text{if } n = 2 \\ T(n-1) + T(n-2) + 4c & \text{if } n > 2 \end{cases},$$

where $c > 0$. Clearly, this recurrence equation can be retrieved from (12) as a particular case when we fix $k = 2$, $a_1 = a_2 = 1$, $c_1 = 2c$, $c_2 = 3c$ and $h(n) = 4c$ for all $n \in \mathbb{N}$. Then, taking

$$\Psi_T(f)(n) = \begin{cases} 2c & \text{if } n = 1 \\ 3c & \text{if } n = 2 \\ f(n-1) + f(n-2) + 4c & \text{if } n > 2 \end{cases},$$

for all $f \in \mathcal{C}_{c,2}$, Theorem 5 guarantees the existence and uniqueness of the solution in $\mathcal{C}_{c,2}$, say f_T^F, to the above recurrence equation in such a way that it matches up with the running time of computing of the recursive algorithm under consideration.

Next, fix $r > 0$ and $\alpha > \frac{1}{2}$ and define the function $h_{\alpha,r}$ by

$$h_{\alpha,r}(n) = \begin{cases} 2c & \text{if } n = 1 \\ 3c & \text{if } n = 2 \\ r\alpha^n & \text{if } n > 2 \end{cases}.$$

Since $\alpha > \frac{1}{2}$, the D'Alembert ratio test guarantees that $h_\alpha \in \mathcal{C}_{c,2}$.

It is not hard to see that Ψ_T is a cont-improver with respect to the complexity function h_α (i.e. $\Psi_T(h_\alpha) \leq h_\alpha$) if and only if $\alpha \geq 1.61$ and $r \geq \frac{4c}{\alpha^2}$. Hence we obtain, by statement 1) in Theorem 7, that the running of Fibonacci holds $f_T^F \in \mathcal{O}(h_{1.61,\frac{4c}{\alpha^2}})$.

Moreover, a straightforward computation shows that at Ψ_T is a worsener with respect to the complexity function $h_{\alpha,r}$ (i.e. $h_{\alpha,r} \leq \Psi_T(h_{\alpha,r})$) if and only if $\alpha \leq 1.61$ (r is not subject to any constraint). Thus we deduce, by statement 2) in Theorem 7, that $f_T^F \in \Omega(h_{1.61,\frac{4c}{\alpha^2}})$.

Therefore we obtain that $f_T^F \in \Theta(h_{1.61,\frac{4c}{\alpha^2}})$, which agrees with the complexity class that can be found in the literature for the recursive algorithm under study, i.e. $f_T^F \in \Theta((1.61)^n)$ (Cull et al., 1985).

5. Asymptotic Complexity Analysis and Denotational Semantics via a common mathematical framework

As we have pointed out in Sections 1 and 3, the main objective of Schellekens' work was to apply the fixed point technique of the Scott approach for Denotational Semantics to Asymptotic Complexity Analysis. After accomplishing the original aim, he suggested the possibility of giving applications of the complexity space framework to new realms of Computer Science as, in particular, to Denotational Semantics. Thus, the research line began by Schellekens would have benefited from the fundamentals of Denotational Semantics (concretely from Scott's ideas) and, at the same time, the latter would benefit from the ideas and techniques that could be originated in the context of the complexity space. In this direction, a few fruitful interactions between both research disciplines have recently been shown in (Romaguera & Valero, 2008), (Rodríguez-López et al., 2008), (Llull-Chavarría & Valero, 2009) , (Romaguera & Valero, 2011b) and (Romaguera et al., 2011a).

In this section we go more deeply into the relationship between both aforenamed disciplines and present a unified mathematical structure that will be helpful in Asymptotic Complexity Analysis and Denotational Semantics simultaneously. For this purpose, consider a recursive algorithm computing the factorial of a nonnegative integer number by means of the recursive denotational specification introduced in Section 1, that is

$$fact(n) = \begin{cases} 1 & \text{if } n = 1 \\ nfact(n-1) & \text{if } n \geq 2 \end{cases}. \tag{14}$$

It is clear that the running time of computing of such a recursive algorithm is the solution to the following recurrence equation (Boxer and Miller, 2005):

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ T(n-1) + d & \text{if } n \geq 2 \end{cases}, \tag{15}$$

where $c, d > 0$. In the light of the preceding recursive specifications, our objective is to be able to describe simultaneously under a unique fixed point technique the running time of computing and the meaning of the recursive algorithm that computes the factorial of a nonnegative integer number, that is to provide the complexity class of the solution to the recurrence equation (15) and to prove that the unique solution to the denotational specification (14) is the entire factorial function. To this end we will need to recall the following facts.

On one hand, in (Romaguera & Schellekens, 2000) S. Romaguera and Schellekens introduced and studied a new complexity space setting, that we will call generalized complexity spaces, whose construction is as follows:

Given a quasi-metric space (X, d) and a fixed $x_0 \in X$, the generalized complexity space of (X, d, x_0) is the quasi-metric space $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$, where

$$\mathcal{C}_{X,x_0} = \{f : \mathbb{N} \longrightarrow X : \sum_{n=1}^{\infty} 2^{-n} d^s(x_0, f(n)) < \infty\},$$

and the quasi-metric $d_{\mathcal{C}_{X,x_0}}$ on $\mathcal{C}_{X,x_0}$ is defined by

$$d_{\mathcal{C}_{X,x_0}}(f,g) = \sum_{n=1}^{\infty} 2^{-n} d\left(f(n), g(n)\right),$$

for all $f, g \in \mathcal{C}_{X,x_0}$.

Notice that if we take in the preceding definition the base quasi-metric space (X,d) with $X = (0,\infty]$, $x_0 = \infty$ and $d = u_{-1}$, then the generalized complexity space $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$ is exactly the original complexity space $(\mathcal{C}, d_{\mathcal{C}})$.

In (Romaguera & Schellekens, 2000) several mathematical properties, which are interesting from a computational point of view, of generalized complexity spaces were studied. Among them we are interested in the bicompleteness, because it will be useful for our aim later on. In particular, we have the following result.

Theorem 8. *Let (X,d) be a quasi-metric space and let $x_0 \in X$. Then the generalized complexity space $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$ is bicomplete if and only if (X,d) is bicomplete .*

On the other hand, in the literature it has been introduced the so-called domain of words as a possible mathematical foundation of Denotational Semantics (Davey & Priestley, 1990; Künzi, 1993; Matthews, 1994). Such a mathematical structure can be constructed as follows:

Denote by $\mathbb{N}^\omega$ the set of all finite and infinite sequences (words) over $\mathbb{N}$. Denote by $\sqsubseteq$ the prefix order on $\mathbb{N}^\omega$, i.e. $x \sqsubseteq y \Leftrightarrow x$ is a prefix of y. The pair $(\mathbb{N}^\omega, \sqsubseteq)$ is known as the domain of words over the alphabet $\mathbb{N}$.

Moreover, for each $x \in \mathbb{N}^\omega$ the length of x will be denoted by $\ell(x)$. Thus, $\ell(x) \in [1,\infty]$ for all $x \in \mathbb{N}^\omega$,. In the following, given $x \in \mathbb{N}^\omega$, we will write $x = x_1, x_2 x_3, \ldots$ whenever $\ell(x) = \infty$ and $x = x_1 x_2, \ldots, x_n$ whenever $\ell(x) = n < \infty$.

According to (Matthews, 1994) and (Künzi, 1993), the domain of words can be endowed with a quasi-metric. Indeed, define on $\mathbb{N}^\omega$ the function $d_{\mathbb{N}^\omega} : \mathbb{N}^\omega \times \mathbb{N}^\omega \to \mathbb{R}^+$ by

$$d_{\mathbb{N}^\omega}(x,y) = 2^{-\ell(x,y)} - 2^{-\ell(x)},$$

for all $x, y \in \mathbb{N}^\omega$, where $\ell(x,y)$ denotes the length of the longest common prefix of x and y provided that such a prefix exists, and $\ell(x,y) = 0$ otherwise. Of course we adopt the convention that $2^{-\infty} = 0$.

It is well known that $(\mathbb{N}^\omega, d_{\mathbb{N}^\omega})$ is a bicomplete quasi-metric space. Furthermore, $x \sqsubseteq y \Leftrightarrow d_{\mathbb{N}^\omega}(x,y) = 0$.

Next we present the new mathematical framework and we apply it to to discuss, in the spirit of Scott and Schellekens, the complexity and the meaning of a recursive algorithm computing the factorial of a nonnegative integer number. First of all, we mix the original complexity space and the domain of words via a generalized complexity space in the following way:

Since $((0,\infty], u_{-1})$ and $(\mathbb{N}^\omega, d_{\mathbb{N}^\omega})$ are bicomplete quasi-metric spaces we have that $((0,\infty] \times \mathbb{N}^\omega, u_{-1} + d_{\mathbb{N}^\omega})$ is a bicomplete quasi-metric space, where $(u_{-1} + d_{\mathbb{N}^\omega})((x_1,x_2),(y_1,y_2)) = u_{-1}(x_1,y_1) + d_{\mathbb{N}^\omega}(x_2,y_2)$, for all $(x_1,x_2),(y_1,y_2) \in (0,\infty] \times \mathbb{N}^\omega$.

Hence $(\mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}, d_{\mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}})$ is, by Theorem 8, a bicomplete quasi-metric space, where by $1_{\mathbb{N}^\omega}$ we denote the word of $\mathbb{N}^\omega$ such that $\ell(1_{\mathbb{N}^\omega}) = 1$ and $(1_{\mathbb{N}^\omega})_1 = 1$.

Note that if $f \in \mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}$, then we can write $f = (f_1, f_2)$, with $f_1 : \mathbb{N} \longrightarrow (0,\infty]$ and $f_2 : \mathbb{N} \longrightarrow \mathbb{N}^\omega$. Moreover $f_1 \in \mathcal{C}$ because

$$\sum_{n=1}^{\infty} 2^{-n} \frac{1}{f_1(n)} \leq \sum_{n=1}^{\infty} 2^{-n} (u_{-1} + d_{\mathbb{N}^\omega})^s((\infty, 1_{\mathbb{N}^\omega}), f(n)) < \infty.$$

In fact $f_1 \in \mathcal{C}_{c,1}$, where $c = f_1(1)$.

Now for $f = (f_1, f_2) \in \mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}$ define

$$\Gamma(f)(n) = (\Psi_T(f_1)(n)), M(f_2(n))),$$

for all $n \in \mathbb{N}$, where, compare Theorem 5, $\Psi_T : \mathcal{C}_{c,1} \longrightarrow \mathcal{C}_{c,1}$ is the functional associated to (15) (given by (13)) and $M : \mathbb{N}^\omega \longrightarrow \mathbb{N}^\omega$ is the mapping defined by

$$(M(x))_n = \begin{cases} 1 & \text{if } n = 1 \\ n x_{n-1} & \text{if } 1 < n \leq \ell(x) + 1 \end{cases}, \tag{16}$$

for all $x \in \mathbb{N}^\omega$.

Observe that $d_{\mathbb{N}^\omega}(1_{\mathbb{N}^\omega}, M(x)) = 0$ and $d_{\mathbb{N}^\omega}(M(x), 1_{\mathbb{N}^\omega}) = 2^{-1} - 2^{-(\ell(x)+1)}$, for all $x \in \mathbb{N}^\omega$.

Then $\Gamma(f) \in \mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}$, because

$$\begin{aligned} &\textstyle\sum_{n=1}^{\infty} 2^{-n} (u_{-1} + d_{\mathbb{N}^\omega})^s((\infty, 1_{\mathbb{N}^\omega}), \Gamma(f)(n)) && = \\ &\textstyle\sum_{n=1}^{\infty} 2^{-n} \left[u_{-1}(\infty, \Psi_T(f_1)(n))) + d_{\mathbb{N}^\omega}(M(f_2(n)), 1_{\mathbb{N}^\omega})\right] && \leq \\ &\textstyle\sum_{n=1}^{\infty} 2^{-n} \max\{\tfrac{1}{\Psi_T(f_1)(n)}, \tfrac{1}{4}\} && < \infty. \end{aligned}$$

Furthermore, it is a simple matter to check that

$$\begin{aligned} &d_{\mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}}(\Gamma(f), \Gamma(g)) && = \\ &\textstyle\sum_{n=1}^{\infty} 2^{-n} \left[u_{-1}(\Psi_T(f_1)(n)), \Psi_T(g_1)(n)) + d_{\mathbb{N}^\omega}(M(f_2(n)), M(g_2(n)))\right] && \leq \\ &\tfrac{1}{2} \textstyle\sum_{n=1}^{\infty} 2^{-n} \left[u_{-1}(f_1(n), g_1(n)) + d_{\mathbb{N}^\omega}(f_2(n), g_2(n))\right] && = \\ &\tfrac{1}{2} d_{\mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}}(f, g), \end{aligned}$$

for all $f, g \in \mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}$.

Therefore, by Theorem 1, we can deduce that the functional Γ has a unique fixed point $f_{fact} = (f_{fact,1}, f_{fact,2}) \in \mathcal{C}_{(0,\infty]\times\mathbb{N}^\omega,(\infty,1_{\mathbb{N}^\omega})}$. Of course, by construction of Γ, we have that $f_{fact,1} \in \mathcal{C}_{c,1}$.

Moreover, $f_{fact}(n) = \Gamma(f_{fact})(n)$ if and only if $f_{fact,1}(n) = \Psi_T(f_{fact,1})(n)$ and $f_{fact,2}(n) = M(f_{fact,2}(n))$ for all $n \in \mathbb{N}$.

Notice that a complexity function (a function belonging to $\mathcal{C}$) is a solution to the recurrence equation (15) if and only if it is a fixed point of the functional Ψ_T and that a word is a fixed point of the mapping M if and only if it satisfies the denotational specification (14). Whence we obtain that $f_{fact,1}$ is the solution to the recurrence equation (15) and that $f_{fact,2}$ is the solution to the recursive denotational specification (14). Moreover, by construction of M, $f_{fact,2}(n) = n!$ for all $n \in \mathbb{N}$. Hence $f_{fact,1}$ represents the running time of computing of the recursive algorithm under study and $f_{fact,2}$ provides the meaning of the recursive algorithm that computes the factorial of a nonnegative number using the denotational specification (14), respectively.

So we have shown that generalized complexity spaces in the sense of (Romaguera & Schellekens, 2000) are useful to describe, at the same time, the complexity and the program correctness of recursive algorithms using recursive denotational specifications. The exposed method is also based on fixed point techniques like the Schellekens and Scott techniques but it presents an advantage with respect the latter ones. Indeed, the method provided by generalized complexity spaces allows to discuss the correctness of recursive algorithms which is an improvement with respect to the Schellekens technique and, in addition, it allows to analyze the correctness of the recursive algorithms by means of quantitative techniques which is an improvement with respect to the classical Scott technique that, as we have pointed out in Section 1, is based only on qualitative reasonings.

6. Conclusions

In 1970, D.S. Scott introduced a mathematical framework as a part of the foundations of Denotational Semantics, based on topological spaces endowed with an order relation that represents the computational information, which allowed to model the meaning of recursive denotational specification by means of qualitative fixed point techniques. Later on, in 1995, M.P. Schellekens showed a connection between Denotational Semantics and Asymptotic Complexity Analysis applying the original Scott ideas to analyze the running time of computing of Divide and Conquer algorithms but, this time, via quantitative fixed point techniques (the topology is induced by a quasi-metric that provides quantitative information about the elements of the mathematical framework). In this chapter, we have extended the Schellekens technique in order to discuss the complexity of recursive algorithms that do not belong to the Divide and Conquer family. In particular, we have introduced a new fixed point technique that allows to obtain the asymptotic complexity behavior of the running time of computing of the aforesaid recursive algorithms. The new technique has the advantage of providing asymptotic upper and lower bounds of the running time of computing while that the Schellekens technique only allows to obtain upper asymptotic bounds. Furthermore, we have gone more deeply into the relationship between Denotational Semantics and Asymptotic Complexity Analysis constructing a mathematical approach which allows, in the spirit of Scott and Schellekens, to model at the same time the running time and the meaning of a recursive algorithm that performs a task using a recursive denotational specification by means of quantitative fixed point techniques and, thus, presents a improvement with respect to the Scott and Schellekens approaches.

7. Acknowledgements

The authors thank the support from the Spanish Ministry of Science and Innovation, grant MTM2009-12872-C02-01.

8. References

Boxer, L. & Miller, R. (2005). *Algorithms sequential and parallel: a unified approach*, Charles River Media, ISBN: 1584504129, Massachusetts.

Brassard, G. & Bratley, P. (1988). *Algorithms: Theory and Practice*, Prentice Hall, ISBN: 9780130232434, New Jersey.

Cull, P.; Flahive, R. & Robson, R. (1985). *Difference equations: from rabbits to chaos*, Springer, ISBN: 9780387232348, New York.

Davey, B.A. & Priestley, H.A. (1990). *Introduction to Lattices and Order*, Cambridge University Press, ISBN: 0521784514, Cambridge.

Ecklund, E.F. Jr.&, Cull. P. (1985). Towers of Hanoi and analysis of algorithms. *The American Mathematical Monthly*, Vol. 92, 407–420.

Künzi, H.P.A. Nonsymmetric topology, *Proceedings of Colloquium on Topology*, pp. 303-338, Szekszárd (Hungary), August 1993.

Llull-Chavarría, J. & Valero, O. (2009). An application of generalized complexity spaces to denotational semantics via domain of words. *Lecture Notes in Computer Science*, Vol. 5457, 530–541.

Matthews, S.G. (1994). Partial metric topology. *Annals of the New York Academy of Sciences*, Vol. 728, 183–197.

Rodríguez-López, J.; Romaguera. S & Valero, O. (2008). Denotational semantics for programming languages, balanced quasi-metrics and fixed points. *International Journal of Computer Mathematics*, Vol. 85, 623–630.

Romaguera, S. & Schellekens, M.P. (1999). Quasi-metric properties of complexity spaces. *Topology and its Applications*, Vol. 98, 311–322.

Romaguera, S. & Schellekens, M.P. (2000). The quasi-metric of complexity convergence. *Quaestiones Mathematicae*, Vol. 23, 359–374.

Romaguera, S.; Schellekens, M.P. & Valero, O. (2011). Complexity spaces as quantitative domains of computation, *Topology and its Applications*, Vol. 158, 853–860.

Romaguera, S.; Schellekens, M.P. & Valero, O. (2011). The complexity space of partial functions: a connection between Complexity Analysis and Denotational Semantics. *International Journal of Computer Mathematics*, Vol. 88, 1819–1829.

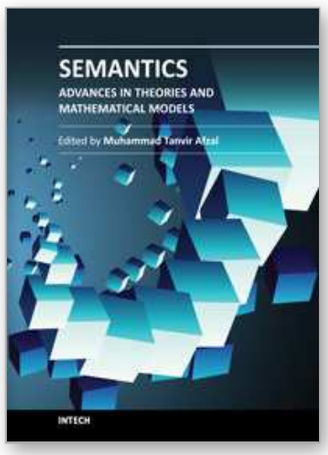
Romaguera, S.; Tirado, P. & Valero, O. (2011). New results on mathematical foundations of asymptotic complexity analysis of algorithms via complexity spaces, *Proceedings of the 11th International Conference on Mathematical Methods in Science and Engineering*, pp. 996-1007, ISBN: 9788461461677, Alicante (Spain), July 2011.

Romaguera, S. & Valero, O. (2008). On the structure of the space of complexity partial functions. *International Journal of Computer Mathematics*, Vol. 85, 631–640.

Schellekens, M.P. (1995). The Smyth completion: a common foundation for denotational semantics and complexity analysis, *Electronic Notes in Theoretical Computer Science*, Vol. 1, 211–232.

Scott, D.S. (1970). Outline of a mathematical theory of computation, *Proceedings of 4th Annual Princeton Conference on Information Science and Systems*, pp. 169-176, Princeton (New York), March 1970.

Scott, D.S. (1982). Domains for denotational semantics. *Lecture Notes in Computer Science*, Vol. 140, 577– 613.



Semantics - Advances in Theories and Mathematical Models

Edited by Dr. Muhammad Tanvir Afzal

ISBN 978-953-51-0535-0
Hard cover, 284 pages
Publisher InTech
Published online 25, April, 2012
Published in print edition April, 2012

The current book is a nice blend of number of great ideas, theories, mathematical models, and practical systems in the domain of Semantics. The book has been divided into two volumes. The current one is the first volume which highlights the advances in theories and mathematical models in the domain of Semantics. This volume has been divided into four sections and ten chapters. The sections include: 1) Background, 2) Queries, Predicates, and Semantic Cache, 3) Algorithms and Logic Programming, and 4) Semantic Web and Interfaces. Authors across the World have contributed to debate on state-of-the-art systems, theories, mathematical models in the domain of Semantics. Subsequently, new theories, mathematical models, and systems have been proposed, developed, and evaluated.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen