Scott’s Qualitative Fixed Point Technique in Complexity Analysis of Algorithms

Maria Lopez-Ramirez, Oscar Valero*

Abstract

In 1972, D.S. Scott developed a qualitative mathematical technique for modeling the meaning of recursive specifications in Denotational Semantics. In this paper we show that the same original Scott’s technique remains helpful for Asymptotic Complexity Analysis of algorithms requiring really a reduced number of hypotheses and elementary arguments. Thus, we will disclose that such a qualitative approach presents a unified mathematical method that is useful for Asymptotic Complexity Analysis and Denotational Semantics. Moreover, we will emphasize the introduced technique applying the results to provide the asymptotic complexity (upper and lower bounds) of the running time of computing of a celebrated algorithm.

2010 AMS Classification: 47H10, 54F05, 68N30, 68Q55, 68Q25, 68W40

Keywords

Partial Order; Scott; Kleene; Fixed Point; Denotational Specification; Algorithmic Complexity; Recurrence Equation

Introduction

In 1972, D.S. Scott developed a mathematical technique for modeling the meaning of recursive specifications in Denotational Semantics. The programming languages used in order to implement such specifications allow, in general, the use of recursive definitions (denotational specifications) in such a way that the meaning of the aforesaid specifications is expressed in terms of its own meaning. Nowadays, the aforementioned mathematical tool is known as fixed point induction principle. Such a principle is based on fixed point theory, the Kleene’s fixed point theorem, for monotone self-mappings defined in partially ordered sets (for a detailed treatment of the topic we refer the reader to [1-4]). Concretely, Scott’s induction principle states that the meaning of a recursive specification is obtained as a fixed point of a non-recursive mapping, induced by the denotational specification, which is the supremum of the successive iterations of the aforesaid non-recursive mapping acting on a distinguished element of the model. The non-recursive mapping expresses the evolution of the program execution. Besides, the partial order encodes some computational information notion so that each successive iteration of the mapping matches up with an element of the mathematical model which is greater than (or equal to) those that are associated to the preceding steps of the program execution. Of course, it is assumed that each iteration of the program computation provides more information about the meaning of the algorithm than those executed before. Therefore, the mentioned fixed point encodes the total information about the meaning provided by the increasing sequence of successive iterations and, in addition, no more information can be extracted by the fixed point than that provided by each element of such a sequence.

The Scott’s fixed point principle have been applied successfully to model computational processes that arise in a natural way in two fields of Computer Science different from Denotational Semantics. Concretely, it has been applied to Asymptotic Complexity Analysis [5] in and to Logic Programming in [6]. In the sequel we focus our attention on the application yielded to Asymptotic Complexity Analysis. In 1995, M.P. Schellekens showed in that the seminal Scott idea of modeling the meaning of a denotational specification as, at the same time, the fixed point of a non-recursive mapping and, in addition, the supremum of the successive iterations sequence can be adapted to Asymptotic Complexity Analysis of algorithms. Thus, Schellekens developed a fixed point technique to get 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 Scott spirit was preserved. The Schellekens technique has a fundamental difference from Scott’s one. Concretely, the fixed point principle is now based on the Banach fixed point theorem instead of Kleene’s fixed point theorem. It must be pointed out that the method of Schellekens introduces a ‘distance’ function, in fact a quasi-metric, which yields a measure of the degree of approximation of the elements that make up the model and, besides, encodes at

Article Information

DOI: 10.31021/acs.20181101
Article Type: Research Article
Journal Type: Open Access
Volume: 1 Issue: 2
Manuscript ID: ACS-1-101
Publisher: Boffin Access Limited

Received Date: 05 September 2017
Accepted Date: 27 December 2017
Published Date: 08 January 2018

*Corresponding author:
Oscar Valero
Department of Mathematical Sciences and Informatics
University of the Balearic Islands
Ctra, Baleares, Spain
E-mail: o.valero@uib.es

Citation: Ramírez ML, Valero O. Scott’s Qualitative Fixed Point Technique in Complexity Analysis of Algorithms. Adv Comput Sci. 2018 Jan; 1(2):101

Copyright: © 2018 Al Valero O, et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution and reproduction in any medium, provided the original author and source are credited.
the same time the information partial order. Moreover, in contrast to
Scott’s qualitative technique, the new method provides a unique fixed
d point of the self-mapping as the candidate running time of computing
of the algorithm (the counterpart of the meaning of the denotational
specification in the Scott case). The quantitative data approach of
the Schellekens technique was seen as an advantage over the Scott
approach and inspired many works on the mathematical foundations
of Asymptotic Complexity Analysis (see, for instance, [7–16]).

Although the preceding quoted works argue that the Schellekens
quantitative approach is very appropriate for Asymptotic Complexity
Analysis and illustrate its strengths, the main target of this paper is
to make a plea for the original Scott fixed point principle showing its
utility for determining the upper and lower asymptotic bounds for
those algorithms whose running time of computing fulfills a general
recurrence equations. Thus, we will show that the Scott approach
remains valid for both, Denotational Semantics and Asymptotic
Complexity Analysis.

The remainder of the paper is organized as follows: In Section
2 we recall brie y the basic notions from asymptotic complexity
of algorithms that we will play a central role in our subsequent
discussion. Section 3 is devoted to showcase the utility of Kleene’s
fixed point theorem, and thus the Scott fixed point approach, for
obtaining the upper and lower bounds of the complexity of those
algorithms whose running time satisfies a general recurrence equation.
All this will be made requiring really a reduced number of
hypotheses and elementary arguments. Finally, we will illustrate the
developed technique applying the results to provide the asymptotic
complexity (upper and lower bounds) of the running time of Quick
sort.

Preliminaries

In this section we recall, on the one hand, the celebrated Kleene
fixed point theorem and the pertinent notions about partial ordered
spaces that are necessary in order to state it. On the other hand, we
remember the mathematical basics about Asymptotic Complexity
Analysis of algorithms that will play a central role in Section 3, where
we will apply the Scott methodology to determine asymptotic bounds
for the running time of computing.

The Kleene fixed point theorem

Following [17] a partially ordered set is a pair (X; ≤ X) such X is a
nonempty set X and ≤ X is a binary relation on X which fulfills for all
x, y, z ∈ X the following:

(i) x ≤ x (reflexivity);
(ii) x ≤ y and y ≤ x ⇒ x = y (anti symmetry);
(iii) x ≤ y and y ≤ z ⇒ x ≤ z (transitivity).

If (X; ≤ X) is a partially ordered set and Y ⊆ X, then an upper bound
for Y in (X; ≤ X) is an element x ∈ X such that y ≤ x for all y ∈ Y . An
element e ∈ Y is least in (Y; ≤ Y ) provided that x ≤ x for all x ∈ Y . Thus,
the supremum for Y in (X; ≤ X), if exists, is an element e ∈ X which is an
upper bound for Y and, in addition, it is the least in the set (UB(Y ); ≤ X),
where UB(Y ) = {x ∈ X: x is an upper bound for Y }. In addition, fixed an
sequence (f(x))x∈X}, the sets {y ∈ X: x ≤ y} and {y ∈ X: x ≤ y} will be denoted by ≤ X
x and by ≤ X x, respectively.

On account of [17] a partially ordered set (X; ≤ X) is said to be chain
complete provided that every increasing sequence has supremum. A
sequence (x)xn is said to be increasing whenever xn ≤ X xn+1 for all n ∈ N,
where N denotes the positive integer numbers set. Moreover, a
mapping from a partially ordered set (X; ≤ X) into itself, we will denote the set (x ∈ X:
f(x) = x)/∈ F (f).

Taking into account the above concepts, Kleene’s fixed point
theorem can be stated as follows (see, for instance, [17]):

Theorem 1: Let (X; ≤ X) be a chain complex partially ordered
set and let f: X → X be an X-continuous mapping. Then the following
assertions hold:

1. If there exists x ∈ X such that x ≤ X f(x), then there exists
   x ∈ F (f) such that x ≤ X x
2. If there exists y ∈ X such that f(y) ≤ X y and
   f(y) ≤ X y, then x ≤ X x

Asymptotic Complexity Analysis

We follow [18] as main reference for Asymptotic Complexity
Analysis of algorithms.

The complexity of an algorithm is determined to be the quantity
of re-sources required by the algorithm to get a solution to the
problem for which it has been designed. A typical resource is the
running time of computing. Usually there are often a few algorithms
that are able to get the solution to a fixed problem. So one of goals is
to set which of them solve the problem taken lower time. With this
aim, it is needed to compare their running time of computing. This
is made by means of the asymptotic analysis in which the running
time of an algorithm is denoted by a function T: N → [0, ∞] such that
T(n) matches with the time taken by the algorithm to solve the
problem, for which it has been designed, when the input data size is
called n. Notice that we exclude 0 from the range of T because
an algorithm always takes an amount of time in order to solve the
problem for which it has been designed.

Since the running time does not only depend on the input data
size n, but it depends also on the particular input and the distribution
of the input data. Hence, it is usually distinguished 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.

Generally, fixed an algorithm, to state the exact expression of the
function which provides its running time of computing is a hard task.
For this reason, in most situations the analysis is focused on bounding
the running time of computing and, thus, to yield an approximation of
it. Of course, to approximate the running time of computing we can
consider an upper and lower bound which are given by the so-called
0 and asymptotic complexity classes. Next we recall such notions.
To this end, from now on, Ω will stand for the usual partial order on
[0, ∞]. In the sequel, we will denote by T the set {f: N → [0, ∞]: f
is monotone with respect to and unbounded. Observe that if a function
f models the running time of computing of an algorithm, then f is
expected to be monotone and unbounded.

Consider two functions f, g ∈ T. Then f ∈ O(f) (f ∈ Ω(g)) if and only
if there exist n0 ∈ N and c ∈ [0, ∞] satisfying f(n) ≤ cg(n) (f(n) ≥ cg(n))
for all n ∈ N with n ≥ n0. Moreover, the case in which f ∈ O(g) ∩
Ω(g) is denoted by f ∈ (g).

Clearly if f ∈ T describes the running time of computing of an
algorithm under discussion, then the fact that f ∈ O(g) gives that
the function g represents an asymptotic upper bound of the running
time. Therefore if we do not know the exact expression of the function
f, then the function g provides an approximate information of the
running time of computing for each input size 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). Clearly, a similar interpretation can be given
when we consider f ∈ (g).

We end the section noting that the set T becomes a partially
ordered set when we endow it with the partial order ≤ T given by f ≤
T g ⇔ f(n) ≥ g(n) for all n ∈ N.
The Scott fixed point technique applied to Asymptotic Complexity Analysis

In this section we show the utility of Kleene’s fixed point theorem, and thus of the Scott fixed point technique, as a mathematical tool for the analysis of the asymptotic complexity of algorithms.

Usually the analysis of the running time of computing of algorithms leads up recurrence equations on N of the following type:

\[ T(n) = \begin{cases} 
\sum_{i=1}^{k} a_i T(n-i) + d(n) & \text{if } n \leq n_0 \\
\sum_{i=1}^{k} a_i T(n-i) + d(n) & \text{if } n > n_0 
\end{cases} \]

(1)

where \( d \in \mathcal{T} \) such that \( d(n) < \infty \) for all \( n \in \mathbb{N}, n_0, k \in \mathbb{N} \) and \( c_i, a_i \in [0, \infty) \), \( \{ \text{for all } i = 1, \ldots, n_0 \} \) and \( d(n) \in [0, \infty) \).

However, a class of recurrence equation that differs from those of type (1) and that arise in a natural way in the analysis of the running time of computing of many Divide and Conquer algorithms are the so-called multi term master recurrences (see, for instance, [19]). The aforementioned recurrence equation is given as follows:

\[ T(n) = \begin{cases} 
\sum_{i=1}^{k} a_i T(n-i) + d(n) & \text{if } n \leq n_0 \\
\sum_{i=1}^{k} a_i T(n-i) + d(n) & \text{if } n > n_0 
\end{cases} \]

(2)

where \( d \in \mathcal{T} \) such that \( d(n) < \infty \) for all \( n \in \mathbb{N}, n_0, k \in \mathbb{N} \) and \( c_i, a_i \in [0, \infty) \), \( \{ \text{for all } i = 1, \ldots, n_0 \} \) and \( d(n) \in [0, \infty) \).

According to [20] the functions d in the preceding expressions are called forcing or input functions.

Examples of algorithms whose running time fulfills a recurrence equation of the above introduced types (either (1) or (2)) are the celebrated Quick sort (worst case), Merge sort (average case), Hanoi Towers Puzzle, Large two (average case) and Fibonacci (see, for instance, [5,18,20,21]). Notice that among the aforementioned algorithms there are recursive and non-recursive.

In the classical literature many techniques have been developed for obtaining the asymptotic bounds of those algorithms whose running time of computing satisfies the preceding recurrence equations. In general, such techniques are specific for each case under study and involve tedious and hard arguments either from mathematical induction or from calculus involving integrals or limits. A general view of the classical treatment of the topic can be found in [18,22].

In what follows we develop a method based on the Kleene fixed point theorem which allows to determine asymptotic bounds in those cases in which the running time of computing satisfies a general recurrence equation which retrieves as a particular case the recurrence equations of type (1) and (2). The new method does not intend to compete with the standard techniques to analyze the complexity of algorithms based on the classical arguments.

The authentic purpose of the novel method is to introduce a formal treatment of asymptotic complexity by means of real basic and elementary arguments which provide, in some sense, a fixed point theoretical counterpart of the classical techniques in the same way that Scott’s fixed point theorem made in Denotational Semantics (see [2-4]). Besides the new method presents the advantage, on the one hand, of avoiding to assume a “convergence condition” in the spirit of Schellekens for all functions involved, that is \( \sum_{i=1}^{k} 2^{-i} \frac{1}{f(n)} < \infty \) (we refer the reader for a detailed treatment of the topic to [5,15]), and, on the other hand, of preserving the Scott spirit and being able to compute, in a natural way and simultaneously, both the meaning and the running time of computing of recursive algorithms.

The technique

Let \( k \in \mathbb{N} \) and let \( g_1 : \mathbb{N} \to \mathbb{N} \) be an unbounded monotone function with respect to the partial order \( \preceq \) such that \( g_i(n) < n \) for all \( i = 1, \ldots, k \).

Fix \( n_0 \in \mathbb{N} \) and denote by \( N_{\infty} \) the set \( \{ n \in \mathbb{N} : n > n_0 \} \). Assume that \( \varphi : N_{\infty} \to [0, \infty) \) is an unbounded monotone function with respect to the partial order \( \preceq \), where \( \preceq \) is defined point-wise, i.e., \( (x_1, \ldots, x_m) \preceq (y_1, \ldots, y_m) \iff x_i \leq y_i \) for all \( i = 1, \ldots, k+1 \).

Next fit \( x_0, \ldots, x_k, c_i, a_i \) \( \in [0, \infty) \). Consider the general recurrence equations on \( N \) given for all \( n \in N \) by

\[ T(n) = \begin{cases} 
c_i T(n) + \varphi(n, T(g_1(n)), \ldots, T(g_k(n))) & \text{if } n \leq n_0 \\
a_i T(n-i) + d(n) & \text{if } n > n_0 
\end{cases} \]

(3)

Clearly the recurrence equations of type (1) and (2) are obtained as a particular case of recurrence equations (3) when

\[ \varphi(n, x_1, \ldots, x_k) = \sum_{i=1}^{k} a_i x_i + d(n), \]

(4)

for all \( n \in N_{\infty} \) and for all \( x_0, x_1, \ldots, x_k \) \( \in [0, \infty) \) [and, in addition, the implicated functions \( g_i \) are chosen respectively as follows:

1. \( g(n) = n \) for all \( n \in N_{\infty} \) and for all \( i = 1, \ldots, k \).

2. \( g(n) = \sum_{i=1}^{k} a_i(n) x_i + d(n) \) for all \( n \in N_{\infty} \) and for all \( i = 1, \ldots, k \).

Another family of recurrence equations that arise in a natural way in the asymptotic analysis of algorithms is the given as follows:

\[ T(n) = \begin{cases} 
c_i T(n) + \varphi(n, x_1, \ldots, x_k) + d(n) & \text{if } n \leq n_0 \\
a_i T(n-i) + d(n) & \text{if } n > n_0 
\end{cases} \]

(5)

where the constants \( a_0, \ldots, a_k \) have been replaced by functions \( a : N_{\infty} \to [0, \infty) \) [for all \( i = 1, \ldots, k \)]. Clearly, the preceding recurrence equations can be retrieved from (3) taking the function as follows:

\[ \varphi(n, x_1, \ldots, x_k) = \sum_{i=1}^{k} a_i(n) x_i + d(n) \]

(6)

for all \( n \in N_{\infty} \) and for all \( x_0, x_1, \ldots, x_k \) \( \in [0, \infty) \) [of course the so-called multi term master recurrences (see, for instance, [19]). The time of computing of many Divide and Conquer algorithms are the type (1) and that arise in a natural way in the analysis of the running time of computing satisfies a recurrence of type (7) is the Quick sort (average behavior). Indeed, its running time is the solution to the recurrence equation given below (see, for instance, [20,22]):

\[ T(n) = \begin{cases} 
\frac{n+1}{n} T(n-1) + 2 & \text{if } n = 1 \\
\frac{n}{n-1} T(n-1) + 2 & \text{if } n > 1 
\end{cases} \]

(8)

With the aim of getting a technique able to provide asymptotic bounds of the running time of computing of algorithms whose analysis yields the preceding recurrence equations, we first stress that every recurrence equation of type (3) has trivially always a unique solution provided the initial conditions \( c_0, \ldots, c_k \) and
taking into account that $\Phi$ and $g_1, \ldots, g_k$ are functions (and thus they give a unique image for a given input). So we only need to focus our attention on how we can get a bound for such a solution without knowing its specific expression. To this end, denote by $\tau_{n, c}$ the subset of $\tau_c$ given by

$$\tau_{n, c} = \{ f \in \tau : f(n) = c_n \text{ for all } n \leq n_0 \}. $$

Now, denote the functional $F_{\Phi}$, $\tau_{n, c} \rightarrow \tau_{n, c}$ by

$$F_{\Phi}(f)(n) = \begin{cases} 
  c_n & \text{if } n \leq n_0 \\
  \Phi(n, f(g_1(n)), \ldots, f(g_k(n))) & \text{if } n > n_0
\end{cases}$$

(9)

for all $f \in \tau_{n, c}$. It is clear that a function belonging to $\tau_{n, c}$ is a solution to the recurrence equation (3) if and only if it is a fixed point of the functional $F_{\Phi}$.

Taking into account the preceding notation we show that the fundamental assumptions that are required by the Kleene fixed point theorem, chain completeness and order-continuity, are satisfied in our approach.

**Proposition 2:** ($\tau_{n, c} \leq \tau_c$) is chain complete.

**Proof:** Let $(f_i)_{i \in I}$ be an increasing sequence in $(\tau_{n, c} \leq \tau)$. Then the function $f \in \tau_{n, c}$ given by

$$f(n) = \sup_{m \in N} f_m(n)$$

for all $n \in N$ is the supremum of $(f_m)_{m \in N}$ in $\tau_{n, c} \leq \tau$. Clearly if $f_m \in \tau_{n, c}$ for all $m \in N$, then $f \in \tau_{n, c}$.\[333\]

**Theorem 3:**

Let $n_0, c \in N$ and let $\Phi$ be the unbounded monotone function associated to (3). If there exists $K \in \mathbb{R}$, such that

$$\Phi(n, x_1 + \varepsilon, \ldots, x_k + \varepsilon) < \Phi(n, x_1, \ldots, x_k) + K \varepsilon$$

(10)

for all $n \in N_{ab}$ and for all $x_1, \ldots, x_k : \varepsilon \in \mathbb{R}$, then $F_{\Phi}$ is strictly monotone and, in addition, they satisfy the regularity condition (10).

**Proof:** Suppose that $(f_m)_{m \in N}$ is an increasing sequence in $(\tau_{n, c} \leq \tau)$. Then Proposition 2 guarantees that the function $f \in \tau_{n, c}$ given by

$$f(n) = \sup_{m \in N} f_m(n)$$

for all $n \in N$ is the supremum of $(f_m)_{m \in N}$ in $\tau_{n, c} \leq \tau$. Next denote the function $f \in \tau_{n, c}$ by

$$f_{\Phi}(n) = \begin{cases} 
  c_n & \text{if } n \leq n_0 \\
  \sup_{m \in \mathbb{N}} \Phi(n, f_m(g_1(n)), \ldots, f_m(g_k(n))) & \text{if } n > n_0
\end{cases}$$

Since $\Phi$ is monotonically with respect to $\leq$ and $f_m \leq \tau$ if for all $m \in N$ we obtain that

$$\Phi(n, f_m(g_1(n)), \ldots, f_m(g_k(n))) \leq \Phi(n, f(g_1(n)), \ldots, f(g_k(n)))$$

for all $n \in N_{ab}$. It follows that

$$f_{\Phi}(n) = \Phi(n, f(g_1(n)), \ldots, f(g_k(n)))$$

(11)

for all $n \in \mathbb{N}_{ab}$. Hence $f_{\Phi} \leq \tau F_{\Phi}(f)$.

It remains to prove that $F_{\Phi}(f) \leq \tau F_{\Phi}(f)$. To this end, we can assume without loss of generality that $f(x_1) < n_0$ for all $n \in N_{ab}$. Since otherwise, by inequality (11), we deduce that $f_{\Phi}(f)(n) = \Phi(n, f(g_1(n)), \ldots, f(g_k(n))) = \infty$. Fix $n \in N_{ab}$.

Then, given $n, \varepsilon \in \mathbb{N}$ such that $f(g_i(n)) < n_0$ for all $i = 1, \ldots, k$.

Hence the monotony of $\Phi$ and (10) give

$$\Phi(n, f(g_1(n)), \ldots, f(g_k(n))) \leq \Phi(n, f_m(g_1(n)), \ldots, f_m(g_k(n))) + \varepsilon$$

(11)

$$\Phi(n, f_m(g_1(n)), \ldots, f_m(g_k(n))) \leq f_{\Phi}(n) + \varepsilon K$$

Whence we deduce that $F_{\Phi}(f)(n) \leq f_{\Phi}(n)$ for all $n \in N_{ab}$ so $F_{\Phi}(f)$ is strictly monotone.

**Example 4:** Fix $n_0, c \in N$ and $c_1, \ldots, c_k \in \mathbb{R}$ such that $c_1 > n_0$ and $c_k > n_0$ and $c_i = 1$ for all $i = 1, \ldots, k$. Consider the function $\Phi : N_{ab} \rightarrow \mathbb{N}$ such that $\Phi(n) = 2^n$ for all $n \in N_{ab}$.

The next example shows that only the monotony of does not provide the $\leq \tau$ -continuity of the function $F_{\Phi}$.

**Example 5:** Fix $n_0, c \in N$ and $c_1, \ldots, c_k \in \mathbb{R}$ such that $c_1 > n_0$ and $c_k > n_0$ and $c_i = 1$ for all $i = 1, \ldots, k$. Consider the function $\Phi : N_{ab} \rightarrow \mathbb{N}$ such that $\Phi(n) = 2^n$ for all $n \in N_{ab}$.

It is obvious that $\Phi$ is monotone with respect to $\leq$ and $\Phi$ does not satisfy the regularity condition (10), since there is not $K \in \mathbb{R}$ such that

$$\Phi(n, x + \varepsilon) \leq \Phi(n, x) + K \varepsilon$$

for all $n \in N_{ab}$ and $x, \varepsilon \in \mathbb{R}$, $\tau$. Indeed, put $\tau = 1$ and take $x = 1/2$. Then

$$\frac{1}{2} + K = \Phi(1, \frac{1}{2}) + K \leq \Phi(1, \frac{3}{2}) = \infty.$$

A straightforward computation shows that $F_{\Phi}$ is not $\leq \tau$-continuous. Clearly the sequence $(f_m)_{m \in N}$ in $\tau_{n, c}$ is given by

$$f_m(n) = \begin{cases} 
  c_n & \text{if } n \leq n_0 \\
  1 - \frac{1}{m + 1} & \text{if } n > n_0
\end{cases}$$

for all $m \in N$, is increasing in $\tau_{n, c}$ and its supremum is the function $f \in \tau_{n, c}$ given by

$$f(n) = \begin{cases} 
  c_n & \text{if } n \leq n_0 \\
  1 & \text{if } n > n_0
\end{cases}$$

Besides, the sequence $(F_{\Phi}(f_m))_{m \in N}$ is increasing in $(\tau, F_{\Phi})$ and its supremum is again the function $f$. Nevertheless, $F_{\Phi}(f) \neq f$.

The next example shows that only the regularity condition (10) for is not enough in order to assure the $\leq \tau$ -continuity of the function $F_{\Phi}$.\[333\]
It is evident that satisfies the regularity condition (10), since

$$\frac{1}{x_1 + x_2 + 2\varepsilon} \leq \frac{1}{x_1 + x_2}$$

for all $n \in N$ and for all $x_1, x_2 \in]0, \infty[$. Moreover, it is clear that $\Phi$ is not monotone with respect to $\leq$. Indeed,

$$\Phi(n, x_1, x_2) = \frac{2}{x_1 + x_2} \geq \frac{1}{x_1 + x_2} = \Phi(n, x_1, x_2)$$

for all $x_1, x_2 \in]0, \infty[$. It follows that $F$ is not monotone with respect to $\leq$ and, thus, it is not $\leq$-continuous.

The next example yields that the $\varepsilon$-continuity of $F$ does not guarantee that the function fulfills the condition (10).

**Example 6:** Fix $n_0 \in N$ and $c_1, \ldots, c_n, g \in]0, \infty[$. Let $g_1, g_2: N \rightarrow N$ be two unbounded monotone functions with respect to $\leq$ such that $g_i(n) < n$ for all $i = 1, 2$. Consider the function $\Phi: N \times ]0, \infty[$ such given by

$$\Phi(n, x_1, x_2) = \begin{cases} \infty & \text{if } x_1 = \infty \text{ or } x_2 = \infty \\ x_1 + x_2 + \varepsilon < K & \text{otherwise} \end{cases}$$

It is evident that the function $F$ is $\leq$-continuous. However, the function $\Phi$ does not satisfy the regularity condition (10). Indeed, assume with the purpose of contradiction that the aforesaid condition is hold. Then, there exists $K \in ]0, \infty[$ such that

$$\Phi(n, x_1 + \varepsilon, x_2 + \varepsilon) < \Phi(n, x_1, x_2) + \varepsilon K$$

for all $n \in N$ and for all $x_1, x_2 \in]0, \infty[$. Set $x_1 = x_2 = K/2$. Then, from the preceding inequality, we deduce that

$$K + \varepsilon < K$$

This is impossible. So, we conclude that does not satisfy the regularity condition (10).

Once we have guaranteed that in our framework the main components required to apply the Kleene fixed point theorem are hold, we introduce the new methodology to provide asymptotic complexity bounds for those algorithms whose running time of computing fulfills a recurrence equation of type (3).

**Theorem 7:** Let $n_0 \in N$ and let $c_1, \ldots, c_n, g \in ]0, \infty[$. Assume that $\Phi: N \times ]0, \infty[$ is an unbounded monotone function with respect to $\leq$ which satisfies the regularity condition (10) and that $f$ is the solution to the recurrence equation of type (3). If there exists $g \in \mathcal{T}_{n_0, c}$, such that $g \leq \mathcal{T}_{n_0, c} F_g(g)$, then $f \in \Omega(g)$. Moreover, if there exists $h \in \mathcal{T}_{n_0, c}$ such that $h \neq \mathcal{T}_{n_0, c} F_g(h)$ and $F_g(h) \sup_{m \in N} h$, then $f \in O(h)$.

**Proof:** First of all, note that $f$ is the unique solution to (3) and, thus, the unique fixed point of $F_f$ in $\mathcal{T}_{n_0, c}$. Suppose that there exists $g \in \mathcal{T}_{n_0, c}$ such that $g \leq \mathcal{T}_{n_0, c} F_g(g)$. Since $\left(\mathcal{T}_{n_0, c}, \leq\right)$ is chain complete and $\Phi$ is $\leq$-continuous we have, by assertion 1) in the statement of Kleene’s fixed point theorem (Theorem 1), that there exists a fixed point $f \in \mathcal{T}_{n_0, c}$ of $F_g$ such that $f \leq \mathcal{T}_{n_0, c} g$ and, hence, that $f \in \Omega(g)$. The fact that $F_g$ admits only a unique fixed point gives that $f = f$ and that $f \in \Omega(g)$.

Suppose that there exists $h \in \mathcal{T}_{n_0, c}$ such that $h \notin \mathcal{T}_{n_0, c} g$ and $F(h) \notin \mathcal{T}_{n_0, c} h$. Assertion 2) in the statement of Theorem 1 provides that $f \leq \mathcal{T}_{n_0, c} h$ Therefore, $f \in O(h)$.

In the light of the preceding result we draw in the inference that in order to get the asymptotic bounds of an algorithm whose running time of computing satisfies a recurrence equation of type (3), it is enough to search the bounds among those functions in $\mathcal{T}_{n_0, c}$ that are “post-fixed” point of $F_g$ and those that are “pre-fixed” point of $F_g$. Moreover, the condition in the statement of Theorem 7 that states a relationship between the post fixed point $g$ and the pre-fixed point $h$, that is $h \neq \mathcal{T}_{n_0, c} g$, points out that really the upper bound class $O(h)$ must be included in the lower bound class $\Omega(g)$ which is reasonable from a complexity theory viewpoint.

**An illustrative example**

Let us consider Quicksort. According to its running time of computing $f_q$ for the average behavior, is the solution to the recurrence equation (12), i.e., to the following recurrence (22):

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

As indicated in Subsection 3.1, the preceding recurrence equation is retrieved from (3) when we consider:

$$k = 1, c_1 = c \text{ and } n_0 = 2.$$ 

$$g(n) = n - 1, a(n) = \frac{n+1}{n} \text{ and } d(n) = 2 \text{ for all } n \in \mathbb{N}_2$$ 

$$\Phi(n, x) = \frac{n+1}{n} x + 2 \text{ for all } n \in \mathbb{N}_2 \text{ and for all } x \in ]0, \infty[.$$ 

$$\Phi(n, \infty) = \infty \text{ for all } n \in \mathbb{N}_2.$$ 

$$f_q(f(n)) = \Phi(n, f(n-1)) = \frac{n+1}{n} f(n-1) + 2 \text{ for all } f \in \mathcal{T}_{n_0, c}, (f \in \mathcal{T}_{n_0, c} f) \text{ and for all } n \in \mathbb{N}_1.$$ 

Notice that $\Phi(n, x + \varepsilon) < \Phi(n, x)$, $K \in \{2, \infty, \cdots, n \in \mathbb{N}_2 \}$, and $x \in ]0, \infty[$.

In the light of the preceding facts we have that all assumptions required in the statement of Theorem 7 are hold. With the aim of making clear the use of the technique yielded by the aforesaid theorem we verify that the asymptotic bounds known in the literature are retrieved from our approach. To this end, let $f_q$ be the solution to the recurrence equation (12).

On the one hand, it is not hard to check that, given $g \in \mathcal{T}_{n_0, c}, g \notin \mathcal{T}_{n_0, c} f_q(g)$ fulfills the following:

$$g(n) \leq 2 + C$$

$$\frac{n+1}{2} + 2(n+1) \sum_{i=1}^{n} \frac{1}{i+2} \text{ if } n \geq 3$$

On the other hand, it is not hard to check that, given $h \in \mathcal{T}_{n_0, c}, F_q(h) \leq h$ fulfills the following:

$$h(n) \geq 2 + \frac{3}{2} C$$

$$\frac{n+1}{2} + 2(n+1) \sum_{i=1}^{n} \frac{1}{i+2} \text{ if } n \geq 3$$

Next take $f \in \mathcal{T}_{n_0, c}$ defined by

$$f(n) = \begin{cases} c & \text{if } n = 1 \\ \frac{c}{2} + \frac{3}{2} C & \text{if } n = 2 \\ \frac{c}{2} + 2(n+1) \sum_{i=1}^{n} \frac{1}{i+2} & \text{if } n \geq 3 \end{cases}$$

By Theorem 7 we deduce that $f_q \in \Omega(f)$ which immediately gives the well-known asymptotic bound $O(f_q)$ (see, for instance, [24]).
where $f_{\log} \in \tau_{\log}$ with

$$f_{\log}(n) = \begin{cases} c & \text{if } n = 1 \\ \frac{3}{2} c + \frac{3}{2} C & \text{if } n = 2 \\ n \log(n) & \text{if } n \geq 3 \end{cases}$$

(13)

We end this subsection pointing out that Scott’s technique presents a unified mathematical approach useful at the same time for asymptotic complexity analysis and denotational semantics. Thus, for instance, such a method allows to model simultaneously both, the meaning and the running time of computing of recursive algorithms using recursive denotational specifications. An easy, but representative, example to which the technique could be applied is given by an algorithm computing the factorial function by means of a recursive specification whose meaning satisfies the following de-notational specification (14) and its running time of computing fulfills the following recurrence equation (15):

$$f_{\text{act}}(n) = \begin{cases} 1 & \text{if } n = 1 \\ n f_{\text{act}}(n-1) & \text{if } n > 1 \end{cases}$$

(14)

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

(15)

where $c, d > 0$.

Conclusion

In 1972, D.S. Scott developed a qualitative mathematical technique for modeling the meaning of recursive algorithms in denotational semantics. We have shown that the same original Scott’s technique remains valid for asymptotic complexity analysis of algorithms. So we have seen that such a qualitative approach presents a unified mathematical method that is useful for asymptotic complexity analysis and denotational semantics. This fact presents an advantage over the quantitative technique, based on Banach’s fixed point theorem, introduced in 1995 by M.P. Schellekens because such a technique has been designed mainly for asymptotic complexity analysis. Moreover, the use of the qualitative approach agrees with the qualitative character of the complexity analysis of algorithms in computer science, where to provide the asymptotic behavior of the running time is more important than to get the exact expression of the running time itself. Further-more, using the qualitative framework we avoid to require the convergence condition assumed by the quantitative Schellekens approach whose unique purpose is to guarantee the soundness of a distance, which provides the quantitative information, but it has not a priori a computational motivation.

Acknowledgement

This research was funded by the Spanish Ministry of Economy and Competitiveness under Grants TIN2014-56381-REDT (LODISCO) and TIN2016-81731-REDT (LODISCO II) and AEI/FEDER, UE funds.

References