Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 109C (2017) 1164–1169

Procedia Computer Science

www.elsevier.com/locate/procedia

International Workshop on Adaptive Technology
(WAT 2017)

Towards performance-focused implementations of adaptive devices

Paulo Roberto Massa Cereda[a,*], João José Neto[a]

[a]*Escola Politécnica, Departamento de Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo*
Av. Prof. Luciano Gualberto, s/n, Travessa 3, 158, CEP: 05508-900 – São Paulo, SP – Brasil

Abstract

This paper presents instrumentation metrics for adaptive rule-driven devices as a means to obtain performance-focused implementations, from the underlying non-adaptive rule-driven device to the adaptive mechanism, as well as discussions regarding the adaptive behaviour and its corresponding operations, from theoretical and practical points of view.

Keywords: adaptive devices, instrumentation, implementation

1. Introduction

Adaptivity is the term used to denote a phenomenon in which a device spontaneously changes its internal behaviour in order to accommodate planned yet unexpected situations; these changes are triggered based solely on the device's own rule set and input stimuli, without any external interference[1,2]. A device is called adaptive if such feature is available to the model as a whole.

Although offering no computational power boost, adaptivity provides mechanisms for expressing abstractions more conveniently[1,3,4]. As a direct consequence, several model improvements are made possible and practically viable, such as complexity reduction[5], problem partitioning[6] and hierarchical solving[7], available at almost no sensible cost to the user[3].

Implementations of adaptive devices greatly vary, as well as the models themselves[3,8,9]. An early work of Cereda and José Neto[3] discussed potential bottlenecks and shortcomings on using common software engineering techniques as a means to implementing programs with adaptive characteristics, with drastic – and sometimes fatal – impacts on performance and stability. In this paper, we aim at extending the discussion to the adaptive mechanism, through fine-grained instrumentation of the adaptive behaviour and its corresponding operations, from theoretical and practical points of view.

Instrumentation is the capability of monitoring and recording a device behaviour and measuring performance during its life cycle[10]. It plays a crucial role in evaluation and testing procedures, as the collected data provide basis for achieving better performance and model improvements[11,12,13]. It is generally advisable to combine different metrics

[*] Corresponding author, +55 11 3091-5402.
E-mail addresses: paulo.cereda@usp.br (Paulo Roberto Massa Cereda)., jjneto@usp.br (João José Neto).

10.1016/j.procs.2017.05.390

in order to obtain a more comprehensive representation of such collected data, in an attempt to reduce bias[14,10]. However, producing traces incurs runtime overhead and therefore may interfere with the device's timing and perturb its behaviour[15], so instrumentation has to be kept to a minimum[16,13,17]. As to trace adaptive devices, we exposed their inner workings to analysis and gathered relevant data on queries and operations.

This paper is organized as follows: Section 2 formally introduces the concept of an adaptive rule-driven device. Section 3 presents presents the instrumentation metrics, as well as implementation aspects. Experiments and discussions are presented in Section 4. Finally, conclusions are presented in Section 5.

2. Adaptive rule-driven devices

This section formally introduces the concept of a general adaptive rule-driven device. It is important to observe that any non-adaptive rule-driven device may be enhanced in order to accommodate an adaptive behaviour while preserving its integrity and original properties[2]. The adaptive mechanism acts as simple extension to the underlying non-adaptive device.

Definition 1 (rule-driven device)**.** A rule-driven device is defined as $ND = (C, NR, S, c_0, A, NA)$, such that ND is a rule-driven device, C is the set of all possible configurations, $c_0 \in C$ is the initial configuration, S is the set of all possible input stimuli, $\epsilon \in S$, $A \subseteq C$ is the subset of all accepting configurations (respectively, $F = C - A$ is the subset of all rejecting configurations), NA is the set of all possible output stimuli of ND as a side effect of rule applications, $\epsilon \in NA$, and NR is the set of rules defining ND as a relation $NR \subseteq C \times S \times C \times NA$. □

Definition 2 (rule)**.** A rule $r \in NR$ is defined as $r = (c_i, s, c_j, z)$, $c_i, c_j \in C$, $s \in S$ and $z \in NA$, indicating that, as response to a stimulus s, r changes the current configuration c_i to c_j, processes s and generates z as output[2]. A rule $r = (c_i, s, c_j, z)$ is said to be compatible with the current configuration c if and only if $c_i = c$ and s is either empty or equals the current input stimulus; in this case, the application of a rule r moves the device to a configuration c_j, denoted by $c_i \Rightarrow^s c_j$, and adds z to the output stream. □

Definition 3 (acceptance of an input stimuli stream by a rule-driven device)**.** An input stimuli stream $w = w_1 w_2 \ldots w_n$, $w_k \in S - \{\epsilon\}$, $k = 1, \ldots, n$, $n \geq 0$, is accepted by a device ND when $c_0 \Rightarrow^{w_1} c_1 \Rightarrow^{w_2} \ldots \Rightarrow^{w_n} c$ (in short, $c_0 \Rightarrow^w c$), and $c \in A$. Respectively, w is rejected by ND when $c \in F$. The language described by a rule-driven device ND is represented by $L(ND) = \{w \in S^* \mid c_0 \Rightarrow^w c, c \in A\}$. □

Definition 4 (adaptive rule-driven device)**.** A rule-driven device $AD = (ND_0, AM)$, such that ND_0 is a device and AM is an adaptive mechanism, is said to be adaptive when, for all operation steps $k \geq 0$ (k is the value of an internal counter T starting in zero and incremented by one each time a non-null adaptive action is executed), AD follows the behaviour of an underlying device ND_k until the start of an operation step $k+1$ triggered by a non-null adaptive action, modifying the current rule set; in short, the execution of a non-null adaptive action in an operation step $k \geq 0$ makes the adaptive device AD evolve from an underlying device ND_k to ND_{k+1}. □

Definition 5 (operation of an adaptive device)**.** An adaptive device AD starts its operation in configuration c_0, with the initial format defined as $AD_0 = (C_0, AR_0, S, c_0, A, NA, BA, AA)$. In step k, an input stimulus move AD to the next configuration and starts the operation step $k+1$ if and only if a non-adaptive rule is executed; thus, being the device AD in step k, with $AD_k = (C_k, AR_k, S, c_k, A, NA, BA, AA)$, the execution of a non-null adaptive action leads to $AD_{k+1} = (C_{k+1}, AR_{k+1}, S, c_{k+1}, A, NA, BA, AA)$, in which $AD = (ND_0, AM)$ is an adaptive device with a starting underlying device ND_0 and an adaptive mechanism AM, ND_k is an underlying device of AD in an operation step k, NR_k is the set of non-adaptive rules of ND_k, C_k is the set of all possible configurations for ND in an operation step k, $c_k \in C_k$ is the starting configuration in an operation step k, S is the set of all possible input stimuli of AD, $A \subseteq C$ is the subset of accepting configurations (respectively, $F = C - A$ is the subset of rejecting configurations), BA and AA are sets of adaptive actions (both containing the null action, $\epsilon \in BA \cap AA$), NA, with $\epsilon \in NA$, is the set of all output stimuli of AD as side effect of rule applications, AR_k is the set of adaptive rules defined as a relation $AR_k \subseteq BA \times C \times S \times C \times NA \times AA$, with AR_0 defining the starting behaviour of AD, AR is the set of all possible adaptive rules for AD, NR is the set of all possible underlying non-adaptive rules of AD, and AM is an adaptive mechanism, $AM \subseteq BA \times NR \times AA$, to be applied in an operation step k for each rule in $NR_k \subseteq NR$. □

Definition 6 (adaptive rules)**.** Adaptive rules $ar \in AR_k$ are defined as $ar = (ba, c_i, s, c_j, z, aa)$ indicating that, as response to an input stimulus $s \in S$, ar initially executes the prior adaptive action $ba \in BA$; the execution of ba is cancelled if this action removes ar from AR_k; otherwise, the underlying non-adaptive rule $nr = (c_i, s, c_j, z)$, $nr \in NR_k$ is applied and, finally, the post adaptive action $aa \in AA$ is applied[2]. □

Definition 7 (adaptive function)**.** Adaptive actions may be defined as abstractions named adaptive functions, similar to function calls in programming languages[2]. The specification of an adaptive function must include the following elements: *(a)* a symbolic name, *(b)* formal parameters which will refer to values supplied as arguments, *(c)* variables which will hold values of applications of elementary adaptive actions of inspection, *(d)* generators that refer to new value references on each usage, and *(e)* the body of the function itself. □

Definition 8 (elementary adaptive actions)**.** Three types of elementary adaptive actions are defined in order to perform tests on the rule set or modify existing rules, namely:

- inspection: the elementary action does not modify the current rule set, but allows inspection on such set and querying rules that match a certain pattern. It employs the form $?\langle\text{pattern}\rangle$.
- removal: the elementary action removes rules that match a certain pattern from the current rule set. It employs the form $-\langle\text{pattern}\rangle$. If no rule matches the pattern, nothing is done.
- insertion: the elementary action adds a rule that match a certain pattern to the rule set. It employs the form $+\langle\text{pattern}\rangle$. If the rule already exists in the rule set, nothing is done.

Such elementary adaptive actions may be used in the body of an adaptive function, including rule patterns that use formal parameters, variables and generators available in the function scope. □

Adaptive devices offer conveniences on a more compact model representation and better organization, as each underlying device covers a specific context at a time[18]. Evolutions reflect device fitting based on history and input stimuli in order to accommodate covered yet unexpected scenarios[19,1].

3. Implementation and instrumentation

When implementing a rule-driven device, it is crucial to design an efficient lookup function, i.e, a function that queries the rule set for entries matching the current configuration and stimulus and returns the corresponding result. There is a plethora of data structures for internally storing rules, from hash tables to linked lists and trees; however, the main issue relies on how efficient is the traverse of such structures.

Definition 9 (lookup score function)**.** Let there be a lookup score function $\Omega \colon ND^{all} \mapsto \mathbb{R}$, where ND^{all} is the enumerable set of all rule-driven devices. This function takes a non-adaptive rule-drive device and returns a lookup score based on the rule set size and the average number of steps need to traverse the corresponding data structure. □

For instance, consider a naïve lookup function implementation using an algorithm that requires checking every element in the rule set NR of ND, regardless of hashing. In this scenario, $\Omega(ND) = |NR|$, i.e, the lookup function will take $|NR|$ units (the rule set size) on each call. In general, it is highly advisable to use traverse algorithms such that $\Omega(ND) \ll |NR|$.

Once the lookup score function is properly defined, it is now possible to calculate the overall score of a rule-driven device given an input stimuli stream. For a non-adaptive rule-driven device, in which the rule set is immutable during the entire life cycle, it suffices multiplying the total number of configuration changes by its corresponding lookup score function in order to obtain the total score.

Definition 10 (input stimuli stream score function of a rule-driven device)**.** Let there be an input stimuli stream score function $\rho \colon ND^{all} \times S^* \mapsto \mathbb{R}$, where ND^{all} is the enumerable set of all rule-driven devices and S^* is the reflexive and transitive closure of all input stimuli, such that $\rho(ND, w) = |\{c_0 \Rightarrow^w c_n\}| \cdot \Omega(ND) + \beta$, with β being a constant. □

For instance, consider two rule-driven devices D_1 and D_2, such that $L(D_1) = L(D_2)$. Given an input stimuli stream $w \in S^*$, $\rho(D_1, w) < \rho(D_2, w)$ means that, although both rule-driven devices recognize the very same sentences, D_1 has a lower overall score and therefore is more efficient regarding implementation (and consequently time wise). Also observe that the overall score is calculated at runtime, as the device changes configurations over input stimuli.

Adaptive rule-driven devices are more challenging, as the underlying device potentially evolves over time, as well as operations performed by the adaptive mechanism in order to accomplish such modifications. A close analysis on the inner workings of elementary adaptive actions is needed, in addition to recalculating the lookup score function on each operation step.

Definition 11 (sequence of elementary adaptive actions)**.** Adaptive actions may be defined as adaptive functions. The function body of an adaptive function $\mathscr{A}$ contains a non-empty sequence $E_{\mathscr{A}}$ of elementary adaptive actions, such that $E_{\mathscr{A}} = \langle e_1, e_2, \dots e_n \rangle$ and $e_i \in E_{\mathscr{A}}$ is an elementary adaptive action. □

Definition 12 (elementary adaptive action type lookup function)**.** Let there be an elementary adaptive action type lookup function $type\colon E \mapsto \{\text{inspection}, \text{removal}, \text{insertion}\}$, where E is the enumerable set of all elementary adaptive actions. This function takes an elementary adaptive action $e \in E$ and returns its corresponding type, namely inspection, removal or insertion. □

Definition 13 (variable set lookup function)**.** Let there be a variable set lookup function $vars\colon E \mapsto 2^V$, where E and V are the enumerable set of all elementary adaptive actions and variables, respectively. This function takes an elementary adaptive action $e \in E$ and returns a subset $v \in 2^V$ containing all variables (potentially none) defined in the rule pattern of e. Observe that a variable itself holds a set of elements. □

The underlying non-adaptive device is already covered, as it is a matter of successively applying the lookup function on each operation step (i.e, changes in the internal counter T indicate an underlying device evolution) and accumulating the partial results. The remainder relates to the execution of adaptive actions, in the form of adaptive functions.

Definition 14 (adaptive action score function)**.** Let there be an adaptive action score function $\Psi\colon BA \cup AA \mapsto \mathbb{R}$, where BA and AA are sets of adaptive actions, $a \in BA \cup AA$ is an adaptive action defined as an adaptive function, $E_a = \langle e_{a,1}, e_{a,2}, \dots e_{a,n} \rangle$ is the sequence of elementary adaptive actions of a, k is the current operation step, ND_k is the underlying device, and γ is a constant, such that:

$$h = \begin{cases} 1 & \text{if } vars(e_1) = \emptyset, \\ \prod_{v \in vars(e_i)} \begin{cases} |v| & \text{if } |v| > 0, \\ 1 & \text{otherwise.} \end{cases} & \text{otherwise.} \end{cases} \tag{1}$$

$$\Psi(a) = \sum_{e_i \in E_a} \begin{cases} \Omega(ND_k) + \gamma & \text{if } type(e_i) = \text{inspection}, \\ \Omega(ND_k) \cdot h + h & \text{if } type(e_i) = \text{insertion}, \\ \Omega(ND_k) \cdot h - h & \text{if } type(e_i) = \text{removal}. \end{cases} \tag{2}$$

This function takes an adaptive action a in the form of an adaptive function and returns a score based on both underlying device lookup and the corresponding sequence of elementary adaptive actions. □

Variables may produce combinatorial results mostly due to their own nature, as they hold entire sets instead of single elements. Such scenarios are highly undesirable and should be avoided, specially if detected in design time.

Definition 15 (input stimuli stream score function for adaptive rule-driven devices)**.** Let there be an input stimuli stream score function $\phi\colon AD^{all} \times S^* \mapsto \mathbb{R}$, where AD^{all} is the enumerable set of all adaptive rule-driven devices and S^* is the reflexive and transitive closure of all input stimuli, C_k is the set of all possible configurations for ND in an operation step k, AR_k is the set of all possible rules for AD, k is the current operation step, ND_k is the underlying device, and δ is a constant, such that

$$h = \{(c_j, w_l, c_o) \mid c_j \Rightarrow^{w_l} c_o, \text{ with } c_j, c_o \in C_i, w_l \in S\} \tag{3}$$

$$b = ba \mid ar_p = (ba_p, c_{p,1}, s_p, c_{p,2}, z_p, aa_p) \in AR_i, ba = ba_p, c_m = c_{p,1}, w_n = s_p, c_r = c_{p,2} \tag{4}$$

$$a = aa \mid ar_q = (ba_q, c_{q,1}, s_q, c_{q,2}, z_q, aa_q) \in AR_i, aa = aa_q, c_m = c_{q,1}, w_n = s_q, c_r = c_{q,2} \tag{5}$$

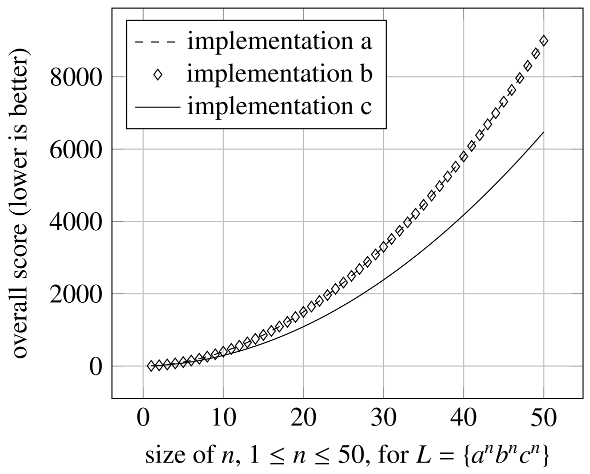
$$\phi(AD, w) = \sum_{i=0}^{k} |h| \cdot \Omega(ND_i) + \sum_{(c_m, w_n, c_r) \in h} \begin{cases} \Psi(b) & \text{if } b \neq \epsilon, \\ 0 & \text{otherwise.} \end{cases} + \begin{cases} \Psi(a) & \text{if } a \neq \epsilon, \\ 0 & \text{otherwise.} \end{cases} + \delta \tag{6}$$

This function takes an adaptive rule-driven device and a input stimuli stream and returns a score based on the underlying device lookup and the adaptive mechanism behaviour. □

The extended input stream score function now calculates the overall score of an adaptive rule-driven device *AD* given an input stimuli stream *w*. Observe that both elements of *AD* are covered, namely the underlying non-adaptive rule-driven device and the adaptive mechanism. Metrics may be collected through a technique known as witness, which consists of writing unique tokens to a trace stream at runtime[12]. Such elements are inserted at points that are targets of predicates (e.g, conditionals).

4. Experiments and discussions

In order to study the behaviour of adaptive devices in a real world scenario, we have modified an open source library for implementing adaptive automata, making it instrumentation-aware. Three implementations of the classical recognizer $M(L)$, such that $L = \{a^n b^n c^n \mid n \in \mathbb{Z}, n \neq 0\}$, were proposed, each one using a different strategy for designing adaptive actions: *(a)* the original formulation as seen in[20], *(b)* actions defined as parameterless functions, and *(c)* actions defined as parametric functions, without elementary adaptive actions of inspection in their bodies. Results are presented in Figure 1. Note that, as the underlying engine used in the three implementations is the same, $\Omega(AD_a) = \Omega(AD_b) = \Omega(AD_c)$. Also, due to space constraints, the implementation details were omitted[1].



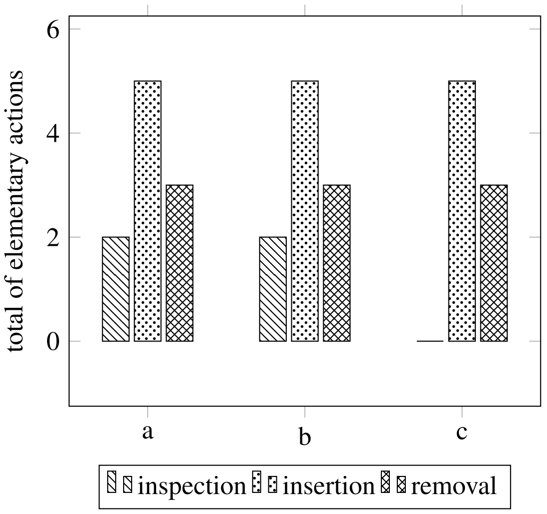


Fig. 1. Overall score and total of elementary actions for three implementations of the classical recognizer $M(L)$, such that $L = \{a^n b^n c^n \mid n \in \mathbb{Z}, n \neq 0\}$, with n ranged from 1 to 50, each one using a different strategy for designing adaptive actions.

According to Figure 1, the number of elementary adaptive actions, as well as how variables are handled inside the function scope, may significantly impact on overall performance. Note that a simple reduction on the number

[1] The experiments are available in the updated library repository: `https://gitub.com/cereda/aa-metrics`.

of elementary adaptive actions offered a performance boost of almost 30% in the long run. However, results are insufficient to pinpoint improvements and shortcomings when using parametric and parameterless functions; our hypothesis is that parameters likely spare inspections in the function body, thus improving performance. Further studies are needed to assess such claim.

Observe that the variable usage was kept to a minimum. It is highly advisable, whenever possible, to avoid inspections which populate variables with multiple values, as elementary adaptive actions performed on variables holding more than one value will cause combinatorial results and consequently a severe performance drop.

5. Conclusions

This paper presented instrumentation metrics for adaptive rule-driven devices as a means to obtain performance-focused implementations. Both elements of the adaptive device were properly covered, namely the underlying non-adaptive rule-driven device and the adaptive mechanism. Additionally, an open source library for implementing adaptive automata was modified in order to become instrumentation-aware and reflect the metrics presented in this paper.

Instrumenting adaptive rule-driven devices allow a better understanding of the inner workings of such devices, as well as particular characteristics of the languages for which they were originally constructed. The overall score gathered from instrumentation provide basis for achieving better implementational performance as well as model improvements. Besides, the instrumentation metrics presented here may be extended in order to cover several domains.

References

1. J. José Neto, Adaptive automata for context -sensitive languages, SIGPLAN Notices 29 (9) (1994) 115–124.
2. J. José Neto, Adaptive rule-driven devices: general formulation and case study, in: International Conference on Implementation and Application of Automata, 2001.
3. P. R. M. Cereda, J. José Neto, Utilizando linguagens de programação orientadas a objetos para codificar programas adaptativos, in: Memórias do IX Workshop de Tecnologia Adaptativa – WTA 2015, São Paulo, 2015, pp. 2–9.
4. H. Pistori, Tecnologia em engenharia de computação: estado da arte e aplicações, PhD thesis, Escola Politécnica, Universidade de São Paulo (2003).
5. R. E. Korf, Planning as search: A quantitative approach, Artificial Intelligence 33 (1) (1987) 65–68.
6. J. Burg, S. Thomas, Computer science: From abstraction to invention, Tech. rep., Department of Computer Science, Wake Forest University (2003).
7. C. A. Knoblock, Learning hierarchies of abstraction spaces, in: Proceedings of the Sixth International Workshop on Machine Learning, Ithaca, NY, USA, 1989, pp. 241–245.
8. Í. S. Vega, Especificações adaptativas e objetos, uma técnica de design de software a partir de statecharts com métodos adaptativos, in: Memórias do VI Workshop de Tecnologia Adaptativa – WTA 2012, 2012.
9. Í. S. Vega, C. E. Camargo, F. S. Marcondes, Elaboração de especificações adaptativas: uma abordagem biomimética, in: Memórias do IX Workshop de Tecnologia Adaptativa – WTA 2015, 2015, pp. 87–90.
10. A. Wert, H. Schulz, C. Heger, R. Farahbod, Generic instrumentation and monitoring description for software performance evaluation, in: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, ACM, New York, NY, USA, 2015, pp. 203–206.
11. W. Paul, J. Vahrenhold, Hunting high and low: Instruments to detect misconceptions related to algorithms and data structures, in: Proceeding of the 44th ACM Technical Symposium on Computer Science Education, ACM, New York, NY, USA, 2013, pp. 29–34.
12. T. Ball, J. R. Larus, Optimally profiling and tracing programs, ACM Transactions on Programming Languages and Systems 16 (4) (1994) 1319–1360.
13. E. Metz, R. Lencevicius, Efficient instrumentation for performance profiling of real-time embedded software systems, Tech. rep. (2003).
14. C. Pavlopoulou, M. Young, Residual test coverage monitoring, in: Proceedings of the 21st International Conference on Software Engineering, ACM, New York, NY, USA, 1999, pp. 277–284.
15. S. Fischmeister, P. Lam, On time-aware instrumentation of programs, in: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, 2009, pp. 305–314.
16. V. Kuncak, P. Lam, K. Zee, M. C. Rinard, Modular pluggable analyses for data structure consistency, IEEE Trans. Softw. Eng. 32 (12) (2006) 988–1005.
17. D. Mahrenholz, O. Spinczyk, W. Schroder-Preikschat, Program instrumentation for debugging and monitoring with aspectc++, in: Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed omputing, 2002.
18. P. R. M. Cereda, J. José Neto, Persistência em dispositivos adaptativos, in: Memórias do VIII Workshop de Tecnologia Adaptativa – WTA 2014, 2014, pp. 120–125.
19. J. José Neto, Um levantamento da evolução da adaptatividade e da tecnologia adaptativa, IEEE Latin America Transactions 5 (2007) 496–505.
20. J. José Neto, Contribuições à metodologia de construção de compiladores, Postdoctoral thesis, Escola Politécnica da Universidade de São Paulo, São Paulo (1993).