An Infinite Proper Subset of Regular Languages as a State Change Based Coupling of Finite Automata

Ahmet Çevik, and Hürevren Kılıç

***Abstract*—We introduce an infinite proper subset of regular languages (RL), so called the state change couple of finite automata (FA). The execution time state change behavior of FA is shown to be modelled as *state automata* (SA). For our purpose, we define a unary operator whose domain is FA and range is SA. The new language class obtained by applying the operator on FA is called *state change languages* (SCL). We show that SCL is closed under union, but not under complementation, homomorphism and inverse homomorphism. We also investigate the properties of SA with empty string transitions. The work given here can be considered as a basis for analyzing the language class properties of the runtime attributes of basic computational models.**

***Index Terms*—Run-time attributes, state change languages, regular languages, finite automata.**

I. Introduction

MODELS of computation are mathematical abstractions of computational devices. They are usually used to study the limits of computation. The same models, on the other hand, can also be used to elaborate on related runtime attributes of computational devices. Specifically, *state change* is such an attribute that can be associated with an execution of any abstract computing model.

State change characterization realized by working on abstract computational models rather than on their implementations gives us: (i) an idea about state change related limits of their implementations under certain conditions (ii) an abstraction to investigate possible "coupled" classes of languages that can be identified through the very basic idea of state changes that occur during the execution of an abstract machine. This in turn may lead to identification of new abstract machines. State change based formalization of abstract machines has been investigated in [1]. Related to the item (ii), search for not coupled but robust classes of regular languages (RL) is known to be a problem of theory of computing, automata theory, formal verification and regular model checking [2], [3]. In [2], for example, a robust decidable class of RL recognized by finite ordered monoids, called W, has been proposed. Other example subclasses of RL identified due to the "negative result that RL cannot be inferred from positive data only" [4] include: k-reversible languages [5], strictly RL [6], regular code languages [7] and uniquely terminating RL [8]. Also note that the work given here should not be confused with algorithmic energy complexity as the latter must take Turing machines as a basis and has been studied for other reasons rather than studying the language properties admitted by the state change characterization. For example, some functions for specific problems that can serve as a model to algorithmic energy complexity of algorithms are introduced in [9], [10] to which we refer the reader for further research regarding algorithmic energy complexity in a broader sense.

In this paper, we propose a simple operator which shall we call the $\mathcal{C}$ *operator* that enables model level state change characterization of its possible implementations. We study a possible "coupled" language class defined by the range of the operator. Here, we restrict ourselves to a unary mapping that operates on state change behavior of finite automata. However, the idea can be extended to higher level abstract machines in the hierarchy in which state change is the common core idea. As a consequence, we identify a new language class what we call *state change languages* (SCL), coupled to RL. Then, we investigate this coupled language class and its closure properties. Note that the effort can clearly be extended to alternative computational models by defining suitable operators in order to observe the corresponding language classes.

The remainder of the paper has the following organization. In Section 2, we define the state change operator, and the language class generated by this. In the same section, we give some basic properties of the operator and show that SCL is a proper subset of RL. In Section 3, we examine the properties of SCL including a discussion on determinism. Section 4 is devoted to the study of SA with ϵ-transitions. The last section includes conclusions and future work.

II. The $\mathcal{C}$ Operator and State Change Languages

A. Notation

First we give the notation. We use standard set theoretical operations $\subseteq$, $\subset$, $\cap$, $\cup$ and $-$ respectively for subset, proper subset, intersection, union and set difference. We denote the empty set by $\emptyset$. For a set A, we denote power set, i.e. the set of all subsets, of A by $\mathcal{P}(A)$. The cardinality of a set A is denoted by $|A|$. For an alphabet Σ, we denote the set of all finite strings over Σ by Σ^*. Similarly, for a natural number k, Σ^k denotes the set of all strings of length k over Σ. We denote the empty string by ϵ.[1] For a language L, we denote the complement of L by $\overline{L} = \Sigma^* - L$. We use lowercase letters such as u, w, v for strings and we use a, b for symbols in the alphabet. For any two languages L and M, we denote the language concatenation by $L \circ M = \{vw : v \in L \text{ and } w \in M\}$.

Definition 1: A *finite automaton* (FA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
Q is a finite set of states,
Σ is the alphabet,

[1] In the figures, we denote the empty string simply by e.

Manuscript received August 9, 2015; revised August 15, 2015.

A. Çevik is with the Department of Philosophy, Middle East Technical University, 06800 Ankara, Turkey. E-mail: a.cevik@hotmail.com

H. Kılıç is with the Department of Computer Engineering, Gediz University, 35665 İzmir, Turkey. Email: hurevren.kilic@gediz.edu.tr

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$\delta : Q \times \Sigma \cup \{\epsilon\} \to \mathcal{P}(Q)$ is the transition function,
$q_0 \in Q$ is the start state, and
$F \subseteq Q$ is the set of final states.

For an automaton A, we denote the language of A by $\mathcal{L}(A)$.

Clearly, the total number of state changes is bounded by the length of the computation. So finding the minimum total number of state changes for an accepting computation is nothing but finding the shortest path from the start state to any of the final states.

B. State Automata and State Change Operator

Definition 2: A *state automaton* (SA) $(Q, \Sigma', \delta', q_0, F)$ is an FA with the following restrictions:

(i) $\Sigma' = \{0, 1\}$ is a restricted two symbol alphabet,
(ii) transition function includes transition rules either of the form
$\delta'(p, 0) = \{p\}$ such that $p \in Q$, or
$\delta'(p, 1) = R$ such that $R \in \mathcal{P}(Q - \{p\})$, or both.

Definition 3: The *state change operator*, denoted by $\mathcal{C}$, is a surjective mapping $\mathcal{C} : FA \to SA$ such that, given an FA $A = (Q_A, \Sigma_A, \delta_A, q_{0_A}, F_A)$, it produces an SA $B = \mathcal{C}(A) = (Q_B, \Sigma_B, \delta_B, q_{0_B}, F_B)$ such that, $Q_B = Q_A$, $\Sigma_B = \{0, 1\}$, $q_{0_B} = q_{0_A}$, $F_B = F_A$, and for each $\delta_A(p, a) = R$, we define δ_B such that

if $p \notin R$, then $\delta_B(p, 1) = R$;
otherwise $\delta_B(p, 0) = \{p\}$, $\delta_B(p, 1) = R - \{p\}$

as the new transition rules.

Simply, the $\mathcal{C}$ operator modifies the transition function of a given FA by mapping its input symbols to either 0 or 1 according to the description given above. The transformation takes $O(|\Sigma||Q|^2)$ steps.

Example 1: Let $A = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_1\})$ be an FA such that δ is defined as in Figure 1.

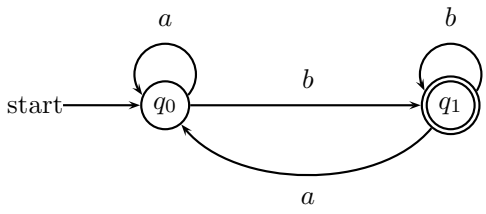

Fig. 1. A finite automaton A.

Notice that in A, state change occurs either when the current state is q_0 and the input is b, or when the current state is q_1 and the input is a. Then, by applying the $\mathcal{C}$ operator, A is transformed by relabeling its input symbols. The transition diagram of the new automaton is given in Figure 2.

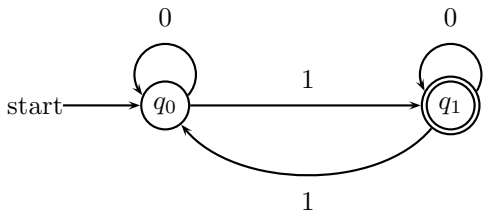

Fig. 2. State automaton $B = \mathcal{C}(A)$.

The output automaton B has a fixed alphabet $\{0, 1\}$. In the transformation, the number of states and the number of transitions are preserved. Since A and B are topologically the same in this sense, we can say that $\mathcal{C}$ is topology preserving. It is also easy to see that $\mathcal{C}$ is idempotent, i.e. $\mathcal{C}(C(A)) = \mathcal{C}(A)$ for any A. Since $\mathcal{C}$ is surjective, it is not invertible. The output automaton describes the state change based coupled automaton description of a given FA under consideration. The rest of the paper will be devoted to the study of the language of this description.

Let Σ and Γ be two alphabets. Recall that a function $f : \Sigma^* \to \Gamma^*$ is a *homomorphism* if $f(\epsilon) = \epsilon$ and $f(w) = f(u)f(v)$ for all $w, u, v \in \Sigma^*$ such that $w = uv$. One can apply a homomorphism to a language by applying it to every string in the language. We observe that the $\mathcal{C}$ operator allows homomorphism of languages only in some cases. To see this let $A = (Q_A, \Sigma_A, \delta_A, q_{0_A}, F_A)$ be an FA without empty string transitions such that $\mathcal{L}(A) \neq \emptyset$. It is easy to see that there exists a homomorphism $h : \Sigma_A^* \to \{0, 1\}^*$ defined over $\mathcal{L}(A)$ and $\mathcal{L}(\mathcal{C}(A))$ iff for each $a \in \Sigma_A$, $\delta_A(p, a) = R$ holds when either $R = \{p\}$ or $R \cap \{p\} = \emptyset$, for all $p \in Q_A$. The general case fails however. Using Example 1 as a counter argument, we show that it is not always the case that homomorphism of languages exists. Suppose for a contradiction that some homomorphism h exists so that whenever $w \in \mathcal{L}(A)$, then $h(w) = h(u)h(v) \in \mathcal{L}(\mathcal{C}(A))$, where $w, u, v \in \Sigma^*$ and $w = uv$. Consider the case $w = bb \in \mathcal{L}(A)$. Then, $h(bb) = h(b)h(b)$ must be in $\mathcal{L}(\mathcal{C}(A))$. However, if we look at Example 1, bb gets the value 10 for $\mathcal{L}(\mathcal{C}(A))$. This means that we cannot assign a unique value for b.

We shall now investigate the language class defined by SA.

Definition 4: A language is a *state change language* (SCL) if some SA recognizes it.

Theorem 1: $SCL \subset RL$.

Proof: First we have to show that every language in SCL is also in RL. This is obvious since every SA is an FA. Next, we have to show that there exists some regular expression E such that $\mathcal{L}(E) \neq \mathcal{L}(B)$ for any SA B. Suppose the contrary. As a counter example, consider the language $\{0\}$ and suppose that $\{0\}$ is in SCL. Then, there must exist an SA B such that $\mathcal{L}(B) = \{0\}$. In this case B must contain a transition rule of the form $\delta(p, 0) = R - \{p\}$ which is necessary for the generation of $\{0\}$, where $R \subseteq Q$ and Q is the set of states of B. But then this contradicts the definition of SA that it cannot include such transitions. ■

In fact, when we think of *state expressions* defining an SA, in the same manner regular expressions defining an FA, we see that any state expression is a regular expression but not the other direction does not always hold. It is easy to see that 0^k cannot be included in any kind of state expression for any $k > 0$.

III. Closure Properties of State Change Languages

It is known that RL is closed under union, intersection, complement, difference, kleene star, concatenation, homomorphism, and inverse homomorphism [12], [11]. SCL is not closed under homomorpism and inverse homomorphism. In fact, the alphabet change is where the problem arises. If we let Γ be the alphabet of an FA such that $|\Gamma| > 2$, then we have a problem of representing state changes. We only have

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

two symbols in SCL. The third symbol is undefined in SCL and this results in any such string, containing three or more different symbols, not being able to be interpreted in the SA since the range of the $\mathcal{C}$ operator is binary. For the similar reason, SCL is not closed under inverse homomorphism $h : \Gamma^* \to \Sigma^*$, for any Γ such that $|\Gamma| > 2$. We next show that SCL is closed under union.

Theorem 2: If L and M are in SCL, then so is $L \cup M$.

Proof: The proof is standard which is by product construction of two SA in this case. If L and M are in SCL, then there exist two SA $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ that recognize L and M, respectively. For $L \cup M$, we construct an SA A such that $\mathcal{L}(A) = L \cup M$. The states of A are of the form (Q_x, Q_y) such that $Q_x \in \mathcal{P}(Q_1)$ and $Q_y \in \mathcal{P}(Q_2)$. Furthermore, the transition function of A will be of the form $\delta_A : Q_1 \times Q_2 \times \Sigma \to \mathcal{P}(Q_1) \times \mathcal{P}(Q_2)$. The start state of A is the pair (q_1, q_2), i.e. the start states of A_1 and A_2. A should accept if and only if either of the automata accepts. Therefore, the final states of A consist of pairs (Q_{xf}, Q_{yf}) such that either $Q_{xf} \in \mathcal{P}(Q_1) \cap F_1 \neq \emptyset$ or $Q_{yf} \in \mathcal{P}(Q_2) \cap F_2 \neq \emptyset$, or both. So, we define $A = (Q_x \times Q_y, \Sigma, \delta_A, (q_1, q_2), Q_{xf} \times Q_{yf})$ and $\delta_A((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$. Thus, $\mathcal{L}(A) = L \cup M$. ■

Another fact about SCL is that it is not closed under complementation unlike RL. Suppose the contrary. Let us again consider the language $L = \{1\}$. Clearly, L is in SCL. Then, for instance, it must be the case that $010 \in \overline{L}$. But any SA which accepts this string must also accept 1 which contradicts the fact that $1 \notin \overline{L}$. So unlike RL, we conclude that SCL is not closed under complementation, homomorphism and inverse homomorphism.

Still further, many languages recognized by SA cannot be recognized by deterministic state automata (DSA). First, let us give the following definition.

Definition 5: An FA $A = (Q, \Sigma, \delta, q_0, F)$ is said to be *incomplete* if $\delta(q, a) = \emptyset$ for some $q \in Q$ and $a \in \Sigma$. Otherwise A is called *complete*.

We argue that there is no general conversion from SA to DSA preserving the language hence observe that SA recognize more languages than DSA. For this we show that there exists a language in SCL which cannot be recognized by any DSA. First let us demonstrate that the application of the standard non-deterministic FA to deterministic FA conversion within the domain of SA fails. A simple case is the language $L = \{1\}$. Although L can be recognized by an SA whose δ consists of a single transition $\delta(q_0, 1) = \{q_f\}$ such that q_0 is the start state and q_f is the final state, this SA is incomplete. Since all deterministic automata are complete, the SA recognizing $\{1\}$ must be non-deterministic. If we define this SA on all arguments to make it complete, then all states must self-loop with a 0 transition which would directly change the language of the automaton. We shall also show that the conversion fails from the domain of FA. Take the same language L. Then we observe that there must be a coupled FA, say A, which, when applied to the C operator, gives an automaton having the language L. If we apply the FA to deterministic FA conversion on A then this will change its topology. But then the obtained deterministic FA will be coupled with an SA whose language is different than L. Also note that given two FA with different languages yet same topology, their coupled SA will be the same.

IV. State Automata with ϵ-Transitions

Forbidding transitions with empty string does put a restriction on constructions for proving closure properties. It naturally arises the question whether or not the empty string in formal language theory provides us a basis to construct the required automata to satisfy more closure properties. Allowing empty string transitions in SA indeed makes things easier and gives more closure properties. If we leave each epsilon transition as it is and so add to the definition of the $\mathcal{C}$ operator an appropriate rule, then some standard constructions lead us to have richer closure properties. Next, we shall investigate the properties of SA when one allows transitions with the empty string, i.e. ϵ-transitions.

By definition, SA do not include ϵ-transitions. On one hand, allowing the $\mathcal{C}$ operator to interpret the empty string like other inputs might sound obscure since there would be "internal" state changes via ϵ-transitions. On the other hand, ϵ-transitions are independent from the input string and this does not violate the notion of state change of an FA for a given particular input string. This motivates us to use ϵ-transitions explicitly in state automata. In this case, we leave each ϵ-transition in a given FA as it is. The definition of state change operator should be therefore extended and adapted for ϵ-transitions.

Definition 6: State change operator with ϵ-transitions, denoted by C_ϵ, is a surjection $C_\epsilon : FA \to SA$ such that, given an FA $A = (Q_A, \Sigma_A, \delta_A, q_{0_A}, F_A)$, it produces an SA with ϵ-transitions $B = C_\epsilon(A) = (Q_B, \Sigma_B, \delta_B, q_{0_B}, F_B)$ such that, $Q_B = Q_A$, $\Sigma_B = \{0, 1\}$, $q_{0_B} = q_{0_A}$, $F_B = F_A$, and for each $\delta_A(p, a) = R$,

$\delta_B(p, \epsilon) = R$ if $(a = \epsilon)$,
$\delta_B(p, 1) = R$ if $(a \neq \epsilon)$ and $(p \notin R)$,
$\delta_B(p, 0) = \{p\}$ and
$\delta_B(p, 1) = R - \{p\}$ if $(a \neq \epsilon)$ and $(p \in R)$.

Let us call ϵ-SCL the set of languages recognized by SA with ϵ-transitions. Clearly, every language in SCL is also in ϵ-SCL. Using ϵ-transitions in SA gives modularity in proving closure properties. Many of them become trivial to construct. Union and product proofs are standard in this case. Other than those two, we give the following closure properties. In fact, notice that it is sufficient to just have SCL in the hypothesis.

Theorem 3: If L and M are in SCL, then $L \circ M$ is in ϵ-SCL.

Proof: Let $S_L = (Q_l, \{0, 1\}, \delta_l, q_{ls}, F_l)$ and let $S_M = (Q_m, \{0, 1\}, \delta_m, q_{ms}, F_m)$ be the SA for L and M, respectively. The standard construction for the concatenation of RL applies here. We construct an SA $S_N = (Q_l \cup Q_m, \{0, 1\}, \delta_n, q_{ls}, F_m)$, where $\{\delta_l\} \cup \{\delta_m\} \subseteq \{\delta_n\}$, with additional ϵ-transitions $\delta_n(q_{lf}, \epsilon) = \{q_{ms}\}$ for each $q_{lf} \in F_l$.

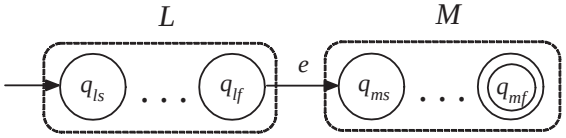

Fig. 3. State automaton S_N with an ϵ-transition recognizing the language $L \circ M$.

As shown in Figure 3, the resulting automaton is an SA with an ϵ-transition. The input string will be accepted if and

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

only if it has an accepting computation on S_N which clearly has the language $L \circ M$. ■

Theorem 4: If L is in SCL, then L^* is in ϵ-SCL.

Proof: Let $S = (Q, \Sigma, \delta, q_0, F)$ be an SA such that $\mathcal{L}(S) = L$, where $F = \{q_{f_1}, \ldots, q_{f_i}\}$ for some $i \geq 1$. We construct an SA with ϵ-transitions $S' = (Q', \Sigma, \delta', q'_s, F')$, as shown in Figure 4 such that $Q' = Q \cup \{q'_s\}$ and $F' = \{q'_s\}$. To obtain δ' so that $\{\delta\} \subset \{\delta'\}$, we add the transition rules

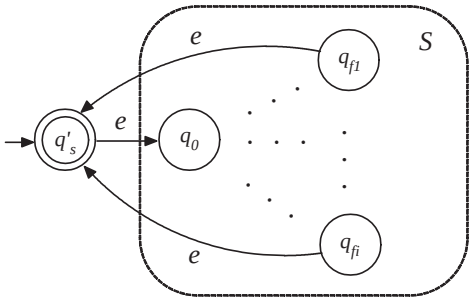

Fig. 4. State automaton S' recognizing the language $(\mathcal{L}(S))^*$.

$\delta'(q'_s, \epsilon) = \{q_0\}$ and $\delta'(q_{f_i}, \epsilon) = \{q'_s\}$ for every $q_{f_i} \in F$. Then, the resulting automaton is clearly an SA with ϵ-transitions such that $\mathcal{L}(S') = L^*$. ■

Let $w = a_1 a_2 \ldots a_n$ be a string. Then, the reversal of w is $w^R = a_n a_{n-1} \ldots a_1$. For example if $w = 00110$, then $w^R = 01100$. We denote the reversal of a language L by $L^R = \{w^R | \ w \in L\}$. Reversal of a transition is simply switching the direction.

Theorem 5: If L is in SCL, then so is L^R.

Proof: Let $A = (Q, \Sigma, \delta, q_0, F)$ be an SA such that $\mathcal{L}(A) = L$. We define $A^R = (Q, \Sigma, \delta^R, s, q_0)$ recognizing L^R such that we define a new start state s with an ϵ-transition to all states in F. Note that we make q_0 as the only accepting state of A^R. Since $\{p\} \in \delta^R(q, a)$ iff $\{q\} \in \delta(p, a)$, we have that $\mathcal{L}(A^R) = (\mathcal{L}(A))^R = L^R$. ■

V. Conclusions

Every state change (or no state change) in an FA is coded as a letter in its coupled SA. We investigate the properties of their state change characteristics through the language class it defines. SCL can be considered as a "coupled" language defined (or coded) by the state transitions of an FA in its computation. The obtained language class has been shown to be closed under union but not under complementation and homomorphism and inverse homomorphism. One may define a language morphism between RL and SCL but only under certain conditions. It has been shown that DSA are less powerful in a sense of language recognition when compared to SA. The properties of SA with ϵ-transitions has been investigated and it has been observed that allowing such transitions gives richer properties with convenient constructions.

From another perspective, introducing the $\mathcal{C}$ operator reveals the fact that there exists at least one interpretation related with the syntax of FA. This interpretation can be taken as the state change attribute.

An analysis through the state change characteristic gives an idea about the relationship between the actual model and its runtime behavior. In our case, the language of this behavor has been shown to be a proper subclass of the language of the actual model. It would be an interesting future work to investigate such coupled language classification for more complex models of computation. For this, one should first introduce a general metric for characterizing the state change of the desired higher model and then observe its language class properties admitted by the metric. Finally, one may then observe how it differs from the one obtained here. Language classification within the state change characteristic of computational models and investigating a possible hierarchy indeed would be an interesting study in its own right. A more interesting study could be done for those abstract models of computation that are not well-defined physically, for example quantum computational models, in order to have an idea about their state change behavior on abstract level.

References

[1] A. Çevik: State Change Languages, Algebraic and Language Class Properties. Atılım University, M.S. Thesis (2009)
[2] A. C. Gómez, J. E. Pin: A Robust Class of Regular languages. MFCS 2008, LNCS 5162, pp. 36-51 (2008)
[3] P. A. Abdulla, B. Jonsson, M. Nilsson, M. Saksena: A survey of regular model checking. International Conference on Concurrency Theory No.15, London, England, vol.3170, pp. 35-48 (2004)
[4] E. M. Gold: Language Identification in the Limit, Inform. Contr., 10,447-474 (1967)
[5] D. Angluin: Inference of Reversible Languages. J. ACM 29, 741-765 (1982)
[6] N. Tanida, T. Yokomori: Polynomial-Time Identifiacation of Strictly Regular Languages in the Limit. IEICE Trans. Inf & Syst. E75-D, 125-132 (1992)
[7] J. D. Emerald, K. G. Subramanian, D. G. Thomas: Learning Code Regular and Code Linear Languages. In: Proceedings of ICGI-96, Lecture Notes in Artificial Intelligence 1147, 211-221 (1996)
[8] E. Mäkinen: Inferring Uniquely Terminating Regular Languages From Positive Data. Information Processing Letters, vol. 62, Issue 2, pp. 57-60 (1997)
[9] R. Jain, D. Molnar, Z. Ramzan: Towards a model of energy complexity for algorithms, Wireless Communications and Networking Conference (WCNC 2005), IEEE, p.1884-1890, Vol.3 (2005)
[10] K. Zotos, A. Litke, A. Chatzigeorgiou, S. Nikolaidis, G. Stephanides: Energy Complexity of Software in Embedded Systems, Nonlinear Sciences - Adaptation and Self-Organizing Systems (2005)
[11] M. Sipser: Introduction to the Theory of Computation, Course Technology, 2nd Ed. (2005)
[12] J. E. Hopcroft, R. Motwani, J. D. Ullman: Introduction to Automata Theory, Languages, and Computation, Addison Wesley, 2nd Ed. (2001)

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)