Computing by Observing: A Brief Survey

Matteo Cavaliere

The Microsoft Research - University of Trento,
Centre for Computational and Systems Biology, CoSBi, Trento, Italy
cavaliere@cosbi.eu

Abstract. This paper is a brief survey of a computational paradigm called *computing by observing* that stresses the role of an observer in computation. The idea of the paradigm is that a computing device can be obtained by combining a basic system and an observer that transforms the trajectories of the basic system into a readable output. The paradigm has been applied in several areas: natural computing (DNA computing and membrane computing), automata and formal language theory. In general, it has been shown that simple basic systems observed by simple observers can produce that which only much more complex systems can produce.

1 Introduction

A standard procedure in many fields of science is to conduct an experiment, observe the entire dynamics of a system and then take the result of this observation as the final output. A momentary picture of the observed system is irrelevant, and rather the entire dynamics of the system is evaluated.

Inspired by this, has been defined a framework where the computation is obtained by observing the entire progress of a basic system. In other words, *the entire observed progress constitutes the result of the computation.* The paradigm (called *computing by observing*) stresses the role of the observer in computation and defines a computation by balancing the power of the observed system and the power of the external observer.

The paradigm, in the most general case, is sketched in Figure 1. Following a set of rules the observer translates the trajectories of the observed basic system into a readable output, usually strings.

In this abstract framework, the basic system has usually a dynamics specified by some fixed rules (applied iteratively) and the observer is a mapping that associates to each configuration reached by the system a label taken from a finite set of labels. Here one considers observed systems and observers as formal machines – the process of observation is actually a process used to filtrate and interpret informations.

The framework was originally [5] applied to a membrane system. Following the same idea, new bio-inspired models of computation have been obtained in [1] (sticker systems) and in [2] (splicing systems). The generalization of the framework to formal language theory has been proposed in [7], to string-rewriting in

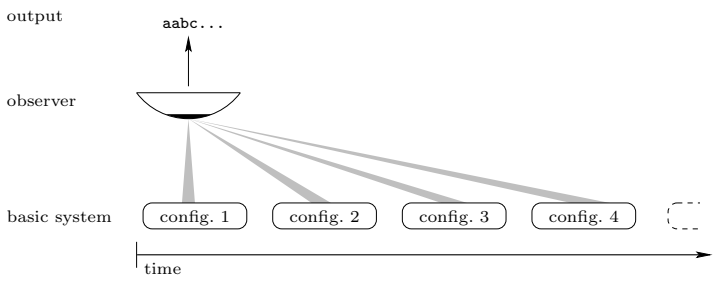

Fig. 1. Basic Idea of the Computing by Observing Framework.

[4, 6], where derivations of grammars and string-rewriting systems are observed by finite state automata. The passage from a basic system as concrete, observable, bio-physical thing to an abstract formal rewriting system can be imagined in the following manner: one can associate a set of (physical) objects to strings and can have some rewriting rules to evolve such objects.

In general, in all these mentioned works, already rather simple basic systems observed by simple observers were proved to be computationally universal (i.e., equivalent to Turing machines).

However, in most of these results it was necessary to design *both* a specific basic system and a specific observer to produce a specific result. On the other hand most basic systems cannot be (easily) programmed or modified (e.g., think to a basic system representing a biological system). However, their evolution can be monitored by using different observers. Then, it is natural to ask how much one can compute by fixing a basic system and choosing different observers (in other words, *computing by only observing*).

This question was (partially) answered in [3] where it was shown that every computational device can be obtained by observing in the "right" manner a *fixed* basic system (with both - observer and observed basic systems with limited computational power).

In this paper we briefly survey the results obtained by using such a paradigm in the formal languages area.

In what follows we use some basic notions and standard terminology from formal language theory (for details, the reader is invited to consult the corresponding chapters of the handbook of formal languages, [9]).

We only recall that $\mathcal{R}EG$ and $\mathcal{C}F$ denotes the classes of regular and context-free grammars, respectively and that by REG, CF and RE we denote the classes of languages generated by regular, context-free, and type-0 grammars, respectively.

2 Computing by Observing

In this section we present *grammar/observer (G/O) systems* that are generative devices based on the framework discussed in the Introduction.

In this case, a formal grammar plays the role of the observed basic system and an automaton – be it a finite automaton or even a Turing machine – plays the role of the observer.

Initially, a formal definition of the observer is given, and then the definition of a G/O system is recalled. Three ways of working for a G/O system (*initial, free, always writing*) are presented. Examples of G/O systems are presented later. This section is essentially based on [7].

2.1 An Example of Observer: Automata with Singular Output

As observer, we essentially use a device mapping arbitrarily long strings into just one singular symbol and we call it automata with singular output.

An automaton with singular output is a tuple $A = (Q, V, \Sigma, q_0, \delta, \sigma)$ with state set Q, input alphabet V, initial state $q_0 \in Q$, and a complete transition function δ as known from conventional finite state automata; further there is the output alphabet Σ and a labeling function $\sigma : Q \mapsto \Sigma \cup \{\lambda\} \cup \{\perp\}$ where λ denotes the empty string and $\perp \notin \Sigma$.

The output of an automaton with singular output is the label of the state in which it halts/stops. For a string $w \in V^*$ and an automaton A we then write $A(w)$ for this output; for a sequence $w_1, \ldots, w_n$ of $n \geq 1$ strings over V^* we write $A(w_1, \ldots, w_n)$ for the string $A(w_1) \cdots A(w_n)$.

The class of all (deterministic) automata with singular output will be denoted by FA_O. Clearly, in the same way observers can be obtained from other classes of automata such as pushdown automata, linear bounded automata or Turing machines.

2.2 G/O Systems

We combine a formal grammar and an observer obtaining the central notion of this section: *G/O system*.

A *G/O system* is a pair $\Omega = (G, A)$ constituted by a generative grammar $G = (N, T, S, P)$ and an automaton with singular output (observer) $A = (Q, V, \Sigma, q_0, \delta, \sigma)$ with output alphabet Σ, which then is also the output alphabet of the entire system Ω. The automaton's input alphabet must be the union of N and T from the grammar to make the desired interaction possible, i.e., $V = N \cup T$.

We distinguish three different modes of generation that define three different models of *G/O systems*:

1. writing a non-empty output in every step (*always writing G/O systems*),
2. writing a non-empty output in every step after an initialization phase of writing only λ (*initial G/O systems*), and
3. changing between empty and non-empty output in an arbitrary manner (*free G/O systems*).

In the case of an *always writing G/O system* Ω the language generated by Ω is

$$L_a(\Omega) = \{A(w_1, w_2, \ldots, w_n) \mid S = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n, \\ w_n \in T^* \text{ and } A(w_i) \neq \lambda, \text{ for all } 1 \leq i \leq n\}.$$

Note that the very first sentential form, which is always the starting symbol, is excluded from the observation (otherwise, all words in $L(\Omega)$ would start with the same letter, if the observer was deterministic). The sentential forms $w_i, 1 \leq i \leq n$, are obtained by applying the productions of G, in the standard sequential way, as usually defined in grammars.

The best way to ensure the last condition, i.e., that λ is never written as output, is of course to define the observer in such a way that it can never produces empty output.

For an *initial G/O system* the output is defined as

$$L_i(\Omega) = \{A(w_0, w_1, \ldots, w_n) \mid S = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n, \ w_n \in T^*, \\ \text{and for all } i \in \{1, \ldots, n\}, A(w_0, w_1, \ldots, w_{i-1}) \neq \lambda \\ \text{implies } A(w_i) \neq \lambda\}.$$

Finally, a *free G/O system* generates a language in the following non-restricted manner:

$$L_f(\Omega) = \{A(w_0, w_1, \ldots, w_n) \mid S = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n, \ w_n \in T^*\}.$$

Thus in all three cases the language contains all those words which the observer can produce during the possible terminating derivations of the underlying grammar. Derivations which do not terminate do not produce a valid output; this means that we only take into account finite words. Of course, by considering the other case of non-terminating derivations the G/O systems could also be used to generate languages of infinite words.

We will actually mainly present the variant where we consider as language produced by Ω:

$$L_{\perp,a}(\Omega) = L_a(\Omega) \cap \Sigma^*.$$

In this way the strings in $L_a(\Omega)$ containing $\perp$ are filtered out and they are not present in $L_{\perp,a}(\Omega)$.

Analogously, the languages $L_{\perp,i}(\Omega)$ and $L_{\perp,f}(\Omega)$ are defined.

For a class $\mathcal{G}$ of grammars and a class $\mathcal{O}$ of observers, $\mathcal{L}_a(\mathcal{G}, \mathcal{O})$, $\mathcal{L}_i(\mathcal{G}, \mathcal{O})$, $\mathcal{L}_f(\mathcal{G}, \mathcal{O})$, $\mathcal{L}_{\perp,a}(\mathcal{G}, \mathcal{O})$, $\mathcal{L}_{\perp,i}(\mathcal{G}, \mathcal{O})$, and $\mathcal{L}_{\perp,f}(\mathcal{G}, \mathcal{O})$ denote the classes of all languages generated by G/O systems with grammars from $\mathcal{G}$ and observers from $\mathcal{O}$ in the respective modes. Quite obviously we obtain for fixed classes of grammars $\mathcal{G}$ and observers $\mathcal{O}$ the inclusions

$$\mathcal{L}_a(\mathcal{G}, \mathcal{O}) \subseteq \mathcal{L}_i(\mathcal{G}, \mathcal{O}) \subseteq \mathcal{L}_f(\mathcal{G}, \mathcal{O})$$

and the same for the variants where the special symbol $\perp$ can be written.

A description of a (free) G/O system $\Omega = (G, A)$ is presented in Figure 2.

For simplicity, in what follows, we present only the mappings that the observers define, without giving a real implementation (in terms of finite automata) for them.

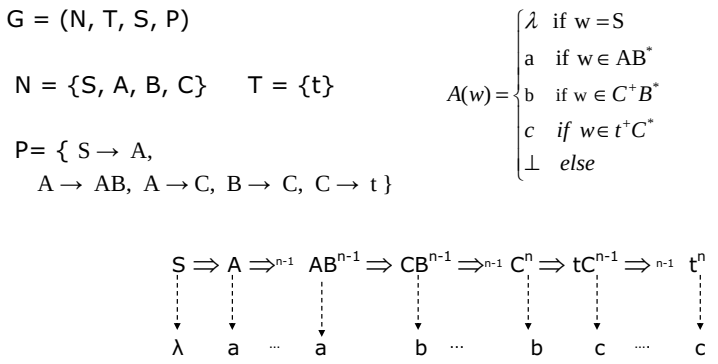

Fig. 2. Consider $\Omega = (G, A)$. At each sentential form produced by the grammar G the observer A associates a symbol that can be $a, b, c, \perp$ or the empty string λ (the vertical arrow is the observer mapping). The concatenation of these symbols is then an output string. For instance, in the figure the output string is $\lambda a \cdots ab \cdots bc \cdots c$. The mapping defined by the observer is specified by the regular expressions. The language $L_f(\Omega)$ is obtained by considering all possible halting derivations of G and collecting all the output strings. Strings in $L_f(\Omega)$ containing $\perp$ are filtered out and not present in $L_{\perp,f}(\Omega)$ that in this case is $\{a^n b^n c^n \mid n > 0\}$.

2.3 Always Writing G/O Systems

The first mode of generation we present is the one of writing an output in every step, i.e., we consider the model of always writing G/O systems. This is maybe the most natural one, since in most cases the observation of an experiment should be complete, at least if about the outcome nothing is known beforehand.

We can see that every context-free language L is in $\mathcal{L}_{\perp,a}(\mathcal{C}F, FA_O)$. For this consider a context-free grammar in the Greibach normal form. There, all right sides of rules are elements of TN^*; this means that in every step exactly one terminal is produced. Since the grammar is still context-free, there is a leftmost derivation for every word in L. In this derivation all sentential forms except the initial S are strings over T^+N^*. An observer can check that a derivation produces only sentential forms of this structure. Then it can output the rightmost terminal for each one of these sentential forms, and the result equals the string derived by the original grammar.

However, $\mathcal{L}_{\perp,a}(\mathcal{C}F, FA_O)$ is bigger than only CF. As an example for a non-context-free language from this class we present $\{a^n b^n c^n \mid n > 0\}$. The grammar

for this language is

$$G = (\{S, A, B, C\}, \{t\}, S, \{S \to A, A \to AB, A \to C, B \to C, C \to t\}).$$

The derivations whose observations will result in the output of words $a^n b^n c^n$ are the ones of the form

$$S \Rightarrow A \overset{n-1}{\Rightarrow} AB^{n-1} \Rightarrow CB^{n-1} \overset{n-1}{\Rightarrow} C^n \Rightarrow tC^{n-1} \overset{n-1}{\Rightarrow} t^n.$$

To produce the output and rule out all other derivations, the observing automaton A will realize the following mapping from the set of sentential forms of G into $\{a, b, c, \perp\}$:

$$A(w) = \begin{cases} a & \text{if } w \in AB^*, \\ b & \text{if } w \in C^+B^*, \\ c & \text{if } w \in t^+C^*, \\ \perp & \text{else.} \end{cases}$$

While $\{a^n b^n c^n \mid n > 0\}$ is still semilinear, also the language $\{a^{2^n} \mid n > 0\}$, which is not semilinear, lies in the class $\mathcal{L}_{\perp,a}(\mathcal{C}F, FA_O)$. To show this, we first recall that a binary tree of depth k has exactly $2^n - 1$ nodes, if the root is considered to already have depth one. Therefore a context-free grammar, which can have full binary trees as derivation trees, and an observer, which can check that only such derivations are made, can generate $\{a^{2^n} \mid n > 0\}$ when interacting in a G/O system. For this example our grammar is $G = (\{S, A, B, C, T_1, T_2, T_3\}, \{t\}, S, P)$, where the set P of productions is

$$\{S \to A, A \to BB, B \to CC, C \to AA, A \to BT_1, B \to CT_2,$$
$$C \to AT_3, A \to t, B \to t, C \to t, T_1 \to t, T_2 \to t, T_3 \to t\}.$$

Now, for example, derivations resulting in the outputs a^2 and a^8 are

$$S \Rightarrow A \Rightarrow t$$

and

$$S \Rightarrow A \Rightarrow BB \Rightarrow CCB \Rightarrow CCCT_2 \Rightarrow tCCT_2 \Rightarrow ttCT_2 \Rightarrow tttT_2 \Rightarrow tttt,$$

respectively. The conditions that the observer has to check (for putting out a in every step) are rather straightforward to see after the previous example.

A sentential form containing any A must be completely changed to one from B^*, the same from B to C, and finally from C to A. This is done by use of the rules $A \to BB, B \to CC$ and $C \to AA$ respectively. To ensure that the entire sentential form is completely changed, the observer maps to a only the sentential forms of the form $A^+B^* \cup B^+C^* \cup C^+A^*$ – others result in the output $\perp$. We notice that there are never more than two different non-terminals present at the same time.

To stop the derivation the rightmost non-terminal of the sentential form must produce the corresponding T_i, with $i \in \{1, 2, 3\}$, by using one of the rules

$A \to BT_1$, $B \to CT_2$, or $C \to AT_3$, and then the only possible further steps are to derive all non-terminals to t. In these cases, the sentential forms mapped to a must be of the form $t^*A^*T_3 \cup t^*B^*T_1 \cup t^*C^*T_2 \cup t^+$.

Therefore the mapping of the observer is

$$A(w) = \begin{cases} a & \text{if} \quad w \in A^+B^* \cup B^+C^* \cup C^+A^* \cup \\ & \qquad t^*A^*T_3 \cup t^*B^*T_1 \cup t^*C^*T_2 \cup t^+, \\ \perp & \text{else.} \end{cases}$$

We note here one difference between the two examples: while in the first case for a word of length n the workspace used by the grammar is $\frac{n}{3}$, in the second case the space is logarithmic in the length of the output.

Trying to characterize the class $\mathcal{L}_{\perp,a}(\mathcal{C}F, FA_O)$ more closely, we can easily see that it is contained in the class of context-sensitive languages. In fact, the G/O system must write a symbol of output at each step and then the total space used by such a system for the generation of a word w is bounded by a constant depending only on the context-free grammar. Therefore, every language generated by an always writing G/O systems is context-sensitive by the workspace theorem. It is an *open problem* whether or not the class $\mathcal{L}_{\perp,a}(\mathcal{C}F, FA_O)$ corresponds exactly to the class of context-sensitive languages.

2.4 Initial G/O Systems

The second variant of G/O systems is called *initial G/O system* and in this model the sentential forms of an initial phase are mapped exclusively to λ. After the first non-empty output only non-empty outputs can be produced – looking back to the biochemical motivation of the concept of evolution and observer, this would correspond to a phase of initializing an experiment and then a phase of actual observation.

Such an initialization phase – which is not restricted, but can be much longer than the actually observed phase – greatly enhances the power of our G/O systems. Indeed, with the same classes of grammars and observers as in Section 2.3 we obtain computational universality in this case.

Theorem 1. $\mathcal{L}_{\perp,i}(\mathcal{C}F, FA_O) = RE$. *[7]*

2.5 Free G/O Systems

An immediate corollary of Theorem 1 is the fact that $\mathcal{L}_{\perp,f}(\mathcal{C}F, FA_O) = RE$. However a G/O system composed of even less powerful components, namely a locally commutative context-free grammar ($\mathcal{L}CCF$) and a finite state automaton is universal, if the output λ can be used without any restriction. Notice that $LCCF$ languages are strictly included in CF languages (e.g., [7]).

Theorem 2. $\mathcal{L}_{\perp,f}(\mathcal{L}CCF, FA_O) = RE$. *[7]*

An *interesting fact* is that intuitively the observer's ability to produce $\perp$, i.e., to eliminate certain computations, seems a powerful and essential feature in all three models. However, we obtain all recursively enumerable languages over Σ simply by intersection of a language over $\Sigma_\perp$ with the regular language Σ^*. Now notice that recursive languages are closed under intersection with regular sets (can be obtained just by looking to the definition of recursive languages). Therefore, there must exist some grammar/observer systems Ω generating a *non-recursive* $\mathcal{L}_f(\Omega)$ (i.e., not using the filtering with $\perp$).

3 Computing by Only Observing

As discussed in the Introduction, it is interesting to understand how much one can compute by allowing only changes in the observer, keeping unchanged the observed basic system.

For this purpose, we say that a grammar is universal (modulo observation) for a family of languages if it can generate every language in the family when observed in an appropriate way.

Formally, as in [3]:

Definition 3. *A grammar G is universal* modulo observation *(m.o.) for a family of languages $\mathcal{L}$ if $\mathcal{L} = \mathcal{L}_{\perp,f}(\{G\}, \mathrm{FA}_O)$.*

We show by means of an example that a G/O system can generate different classes of languages if the observer is changed while the grammar remains fixed.

Let us consider the following context-free grammar:

$$G = (\{S, A, B, C\}, \{t, p\}, S, \{S \to pS, S \to p,\\ S \to A, A \to AB, A \to C, B \to C, C \to t\}).$$

If G is coupled with the observer A' such that $A'(w) = a$ if $w \in \{S, A, B, C, t, p\}^+$, then $\Omega = (G, A')$ defines the language $L_{\perp,f}(\Omega) = \{a^i \mid i \geq 2\}$, a regular language. In fact, the derivation $S \to pS \overset{n-2}{\Rightarrow} p^{n-1}S \to p^n$ produces (when observed) the string a^{n+1}.

Keeping the same grammar G we change the observer into A'' such that:

$$A''(w) = \begin{cases} \lambda & \text{if } w = S, \\ a & \text{if } w \in AB^*, \\ b & \text{if } w \in C^+B^*, \\ c & \text{if } w \in t^+C^*, \\ \perp & \text{else} \end{cases}$$

In this case, one can verifies that $\Omega = (G, A'')$ generates the language $L_{\perp,f}(\Omega) = \{a^n b^n c^n \mid n > 0\}$, a context-sensitive language.

This example suffices to underline that "part" of the computation can be done by choosing the right observer, keeping unchanged the underlying basic system.

Actually, this "part" can be really crucial: in fact, starting from universal type-0 grammars, one can construct an "universal" context-free grammar that can generate all recursively enumerable languages when observed in the correct manner. Formally,

Theorem 4. *There exists a context-free grammar that is universal m.o. for* RE. *[3].*

4 Restrictions on the Observed System

We can consider two restrictions on the observed systems and precisely on the considered context-free grammars: bounding the number of nonterminals and considering leftmost derivations only. In these cases we observe REG and CF, respectively.

The context-free grammar used as a universal grammar in Theorem 4 has no bound on the number of nonterminals in its sentential forms.

The next result shows that indeed this is a necessary property of context-free grammars that are observationally complete for type-0 grammars. In fact, when a bound is imposed, our observations constitute regular languages only. Recall that a context-free grammar is *nonterminal bounded* if there exists a constant k such that all sentential forms generated by the grammar have at most k nonterminals.

Theorem 5. *For every G/O system $\Omega = (G, A)$ with G nonterminal bounded context-free, $A \in FA_O$, $L_{\perp,f}(\Omega)$ is regular. [3]*

A *leftmost* generative grammar is a quadruple $G = (N, T, S, P)$ where G has only leftmost derivations: we assume that $P \subseteq N^+ \times (N \cup T)^*$, and $\alpha \to \beta \in P$ implies $w\alpha x \Rightarrow w\beta x$ for $w \in T^*$ and $x \in (N \cup T)^*$.

A pushdown automaton corresponds to leftmost derivations in a type 0 grammar with productions of the form $pA \to aq\alpha$ for the instruction from state p to q, reading a, popping A and pushing α back to the stack. The following result implies that pushdown automata are less computationally powerful than context-free grammars when observed.

Lemma 6. *For every G/O system $\Omega = (G, A)$ with G a leftmost type-0 grammar, $A \in FA_O$, $L_{\perp,f}(\Omega)$ is context-free. [3]*

This result makes explicit the fact that, when considering this framework, the complexity of the produced output is influenced by the particular dynamics of the observed system.

5 Final Remarks

The survey has tried to show that a relevant parameter in defining the complexity of a computation can be the external observer. Theorem 4 proves that the

observer can be crucial, if the observed system is sufficiently complex: it is possible to 'observe' every recursively enumerable language from a fixed context-free grammar.

It would be then extremely interesting to find grammars (in general systems) that can characterise (by changing the observer) families of languages with certain specified properties. Other directions of investigations are possible: for instance, what classes can be obtained without making use of the symbol $\perp$ (largely used in most of the proofs of the results presented here), or with G/O systems using weaker observers (considering interesting restrictions on finite state automata). On this respect, which are the realistic limitations one should impose on the observer? An interesting suggestion, would be to impose a bound on the time (window) of observation. Another interesting new topic concerns the possibility of using such framework to learn in an easier way complex languages ([8]). Along these lines, we expect several interesting results and, maybe, being a new research area, new type of questions.

References

1. A. Alhazov and M. Cavaliere. Computing by observing bio-systems: The case of sticker systems. *10^{th} International Workshop on DNA Computing*, *LNCS* 3384, Springer, 2004.
2. M. Cavaliere, N. Jonoska, and P. Leupold. DNA splicing: Computing by observing. *Natural Computing. To Appear.*
3. M. Cavaliere, H.J. Hoogeboom, P. Frisco. Computing by only observing. *Proceedings Developments in Language Theory 2006, DLT 2006, LNCS 4036, Springer, 2006.*
4. M. Cavaliere and P. Leupold. Observation of string-rewriting systems. *Fundamenta Informaticae*, 74, 4, 2006.
5. M. Cavaliere and P. Leupold. Evolution and observation - A new way to look at membrane systems. *Workshop on Membrane Computing*, *LNCS* 2933, Springer, 2003.
6. M. Cavaliere and P. Leupold. Evolution and observation - A non-standard way to accept formal languages. *Machines, Computations, and Universality*, *LNCS* 3354, Springer, 2004.
7. M. Cavaliere and P. Leupold. Evolution and observation: A non-standard way to generate formal languages. *Theoretical Computer Science*, 321, 2004.
8. C. Fernando, M. Cavaliere, O. Soyer and R. Goldstein, Hebbian Learning of Context-Sensitive Languages. Manuscript, 2008.
9. G. Rozenberg and A. Salomaa. *Handbook of Formal Languages.* Springer-Verlag, 1996.