A graphic language based on timing diagrams

Christian Antoine, Bernard Le Goff and Jean-Eric Pin

Bull Research and Advanced Development, Rue Jean-Jaurès
78340 Les Clayes-sous-Bois, France

Abstract. We present a new graphic language which can serve, for instance, as models for VLSI and control systems. Its primitives are standard timing diagrams, and this is a great advantage on other formalisms since the designers can rapidly master it. The semantics is rigorously defined in the formalism of the theory of automata on infinite words. Using this formalism, we are able to give a rather precise bound on the expressive power of our graphic language in terms of a language theoretic measure, the *concatenation level.*

1. Introduction

This paper emerged as the result of an enlightening discussion between circuit designers and researchers working in the area of specification languages on the one hand and automata theory on the other hand. It has a practical component, the description of new specification language, but also a strong theoretical flavour, since the semantics of the language is based on recent results of the theory of automata on infinite words.

The origin of our project was the following observation : circuit designers are often discouraged by the complexity of the specification languages. To remedy this problem, we defined a graphic language, called the *Chronogram Language* [1], the primitives of which are standard timing diagrams. Timing diagrams are a formalism which is commonly used in the community of circuit designers, so our language can be rapidly mastered. In other words, contrary to most formalisms, properties are *drawn* rather than written, and this pictural representation is much more convenient for the non-specialist than an abstract formalism.

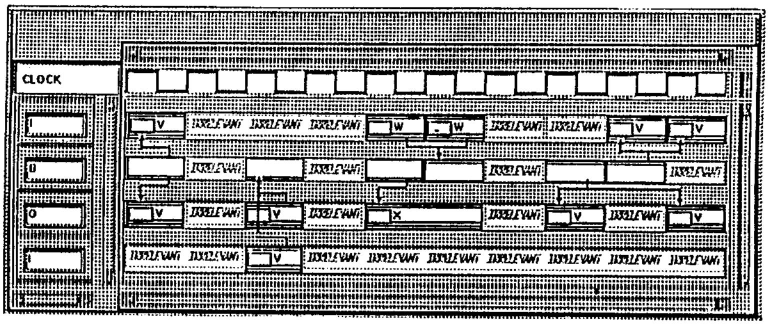
On the other hand, using pictures doesn't prevent one's to have a precise syntax and semantics. This is the place where automata pop up. It turned out that the primitives of our language can be conveniently interpreted by rational (also called *regular*) expressions on infinite words. It follows that *all* chronograms can be interpreted by rational expressions. This approach not only permitted us to define rigorously the semantics of the chronogram language, but also gave some very precise informations on the expressive power of the language. Before we go further into the formulation of our results, we need to briefly review some facts on rational sets of infinite words.

There are essentially two known scales to measure the complexity of a rational set, the logical scale and the combinatorial scale. The logical scale branches into two main parts, corresponding to the first order logic and to the monadic second order logic, respectively. Next one can define a hierarchy inside the first order logic by counting the number of alternations between existential and universal quantifiers. The combinatorial scale takes in account the basic operations that define the rational sets : boolean operations, concatenation product and iteration. It also branches into two main domains : the star-free sets (which can be defined by without iteration) and the rational sets. A hierarchy inside the star-free sets is obtained by counting the number of alternations between the use of the boolean operations and of the concatenation product. A nice (but non-trivial) feature is that the logical and the combinatorial scales are exactly the same [18,15]. Now, our main result states that the languages corresponding to chronograms are of level 3 in the star-free (or logical) hierarchy. This gives a rather precise upper bound to the expressive power of the Chronogram Language.

Although our language was originally designed as a specification language for circuits, it can serve more generally for modelling temporal properties. The Chronogram Language has been designed to provide designers with a good expressive power for temporal properties. For instance, both safety and liveness properties can be expressed into the Chronogram Language, in contrast with other languages VHDL [10], Lucid [2], Lustre [3], Signal [9], etc. in which liveness properties cannot be written. To ensure some compatibility with other existing formalisms, the chronograms that represent safety properties can be compiled into VHDL (a standard description language in the world of circuit designers) and Signal expressions, and liveness properties will be converted into CTL* in the future.

The paper is organized as follows. The Chronogram language is first presented on an example. The abstract syntax of the Chronogram Language is given in section 3. In order to keep the paper self-contained, the main definitions on languages and automata required for this paper are summarized in section 4. The semantics of the Chronogram Language is presented in section 5. The paper ends with a short conclusion section.

2. A presentation of the Chronogram Language.



A chronogram

At the top of this chronogram is defined the CLOCK, which informally, represents the time. All the events are synchronized on the rising edges of the clock. This chronogram defines constraints on the events of three boolean signals: I, O and B. Each line is dedicated to a signal: the second one for B, the third one for O and the first and the last ones for I. Each line consists of *IRRELEVANT* zones and bold line boxes. Only the bold line boxes are relevant for the definition of constraints. On the second line (dedicated to the B signal), there are three boxes with a solid line at the bottom, and three boxes with a solid line at the top. This means that B must carry the false value during the period of time represented by the first three boxes, and the true value during the period of time represented by the last three boxes. On the first and third line, the boxes are labelled by a symbol (v, x or w). This means that during the period of time represented by the box, the signal carries the value v (resp. x or w). This value v (resp. x, w) is not specified in the chronogram but has to be the same in all boxes labelled by v (resp. x, w). A minus sign can be added in the left part of the box: in this case, the signal carries the value $\bar{v}$ opposite to the label v of the box. On the first line such a box is used with the symbol w.

The bold line boxes can be connected by arrows. The resulting graph can have several (simply) connected components. Each component defines a constraint. The relative location of the boxes is relevant only inside a connected component. For instance, the first property specified by the chronogram states: *if B carries the false value, then I and O carry the same value* (denoted by the symbol v in the chronogram). The other properties can be read in a similar way in the chronogram: two linked arrows must be interpreted as a logical and.

3. An abstract syntax of the Chronogram Language

The abstract syntax of the language is given by a grammar, in which the initial of each non-terminal is a capital letter (e.g. `Clock`) and each terminal is either written in capital letters (i.g. `IDENTIFIER`), or consists of a single lower-case letter (e.g. `i`) or of a non alphabetic sign (e.g. `1`, `*`).

The rules are grouped by level of derivation and every rule is written only once. As a consequence, the derivation rules of some terms may precede some of their occurrences.

```
Constraint ::= Property Constraint | Property

Property ::= Clock Hypothesis Conclusion

Hypothesis ::= TimeDiagram    Conclusion ::= TimeDiagram

TimeDiagram ::= MultiColumnList

Clock ::= IDENTIFIER
MultiColumnList ::= MultiColumn MultiColumnList | MultiColumn

MultiColumn ::= StaticMultiColumn | DynamicMultiColumn

StaticMultiColumn ::= Width StaticRowList
DynamicMultiColumn ::= FiniteLowerBound UpperBound DynamicRowList
```

```
StaticRowList ::= StaticRow StaticRowList | StaticRow
DynamicRowList ::= DynamicRow DynamicRowList | DynamicRow

StaticRow ::= StaticIntervalList IDENTIFIER
DynamicRow ::= StaticIntervalList IDENTIFIER

UpperBound ::= FiniteUpperBound | *
Width ::= INTEGER    Length ::= INTEGER
FiniteLowerBound ::= INTEGER    FiniteUpperBound ::= INTEGER

StaticIntervalList ::= StaticInterval StaticIntervalList | NIL
DynamicIntervalList ::= DynamicInterval DynamicIntervalList | NIL

StaticInterval ::= Length PrimitiveSymbol
DynamicInterval ::= PrimitiveSymbol

PrimitiveSymbol ::= i | f | e | r | s | 1 | 0 | SymbolicValue

SymbolicValue ::= - IDENTIFIER | IDENTIFIER
```

4. Languages and automata

In this section, we briefly recall the basic definitions of the theory of automata needed in this article. For more details, the reader is referred to [6,16,19]. We also define the language-theoretic hierarchy that will serve as a measure of the expressive power of the Chronogram Language.

We denote respectively by A^*, A^+ and A^ω the sets of finite words, non-empty finite words and infinite words on an alphabet A. A *language* is a set of finite words, that is, a subset of A^*. The *rational operations* are the three operations union, product and star. The set of rational (or regular) languages of A^* is the smallest set of subsets of A^* containing the finite sets and closed under finite union, product and star. For instance, $\{a, ab\}^*ab \cup (ba^*b)^*$ denotes a rational set. It is possible to generalize the concept of rational languages to infinite words as follows. First, the product can be extended to $A^* \times A^\omega$, by setting, for $X \subset A^*$ and $Y \subset A^\omega$,

$$XY = \{xy \mid x \in X \text{ and } y \in Y\}.$$

Next, we define an infinite iteration ω by setting, for every subset X of A^+

$$X^\omega = \{x_0x_1x_2 \cdots \mid \text{ for all } i \geq 0, x_i \in X\}$$

Equivalently, X^ω is the set of infinite words obtained by concatenating an infinite sequence of words of X. By definition, a subset of A^ω is *ω-rational* (or *ω-regular*) if it is equal to a finite union of sets of the form XY^ω where X and Y are non-empty rational sets of A^+.

Boolean operations comprise union, intersection, complementation and set difference. It can be shown that the rational subsets of A^* are closed under finite boolean operations. The set of *star-free* subsets of A^* is the smallest set of subsets of A^* containing the finite sets and closed under finite boolean operations and product. The set of star-free subsets of A^ω is the smallest set $\mathcal{S}$ of subsets of A^ω closed under finite boolean operations and such that if X is a star-free subset of A^+ and $Y \in \mathcal{S}$, then $XY \in \mathcal{S}$.

The definition of star-free languages of A^* makes use of two different types of operations: boolean operations and concatenation product. By alternating the use of these two operations, one gets a hierarchy, called the *concatenation hierarchy*, defined as follows.

(1) The sets of level 0 are the empty set and A^*,
(2) For every integer $n \geq 0$, the sets of level $n + 1/2$ are the finite unions of the sets of the form $L_0 a_1 L_1 a_2 \cdots a_k L_k$ where L_0, L_1, ..., L_k are sets of level n and a_1, ..., a_k are letters
(3) For every integer $n \geq 0$, the sets of level $n+1$ are finite boolean combinations of sets of level $n + 1/2$.

Note that a set of level m is also a set of level n for every $n \geq m$. Concatenation hierarchies can be extended to infinite words as follows.

(1) The sets of level 0 are the empty set $\emptyset$ and A^ω,
(2) For every integer $n \geq 0$, the sets of level $n + 1/2$ are the finite unions of the sets of the form XaY, where X is a set of A^* of level $n + 1/2$, Y is a subset of A^ω of level n and a is a letter.
(3) For every $n \geq 0$, the sets of level $n + 1$ are finite boolean combinations of sets of level $n + 1/2$.

5. Semantics of the Chronogram Language

The formal semantics of the Chronogram Language comes from ω-rational languages. More precisely, some rational language is associated with each of the graphic primitives of the Chronogram Language and with each variable. Next, to each operator of the Chronogram Language (generation of intervals, rows, columns, multicolumns, time diagrams, etc.) corresponds an operation on languages that preserve rationality. A remarkable feature of the Chronogram Language is the use of *symbolic values* or *boolean variables*. We shall first detail this peculiar aspect.

5.1. Boolean variables and valuations

If v denotes a boolean variable, $\bar{v}$ will denote the boolean variable defined by

$$\bar{v} = \begin{cases} 1 & \text{if } v = 0 \\ 0 & \text{otherwise} \end{cases}$$

Thanks to boolean variables, one can specify in the Chronogram Language not only properties like "The value of the signal at time t is 0 (resp. 1)", but also properties of the form "the value of the signal is v at time t and $\bar{v}$ at time $t + 3$". In order to take in account these variables, it is convenient, in the first place, to represent a signal not as an infinite word on the alphabet $B = \{0, 1\}$, but as an infinite word on the extended alphabet $C = B \cup V \cup \bar{V}$, where V is the set of variables used in the chronogram.

One goes back to the binary alphabet B by associating a value with each variable. This can formally be realized by a *valuation*, that is a map $\nu : C \rightarrow B$ such that for all $b \in B$, $\nu(b) = b$ and for all $v \in V$, $\nu(\bar{v}) = \overline{\nu(v)}$.

For instance, the previous example would be interpreted as "the value of the signal is 0 at time t and 1 at time $t+3$" (which corresponds to the valuation ν defined by $\nu(v) = 0$) or "the value of the signal is 1 at time t and 0 at time $t+3$" (which corresponds to the valuation ν defined by $\nu(v) = 1$).

A valuation $\nu : C \to B$ defines in a natural way a function $\nu : C^* \to B^*$, by setting, for every word $c_1c_2 \cdots c_n \in C^*$, $\nu(c_1c_2 \cdots c_n) = \nu(c_1)\nu(c_2) \cdots \nu(c_n)$. If L is a subset of C^*, the set $\nu(L)$ is called the valuation of L.

5.2. Constraints on a single signal

A signal is considered as an infinite word u on the binary alphabet B. As we shall see later, the constraints defined on a given signal in our language can always be formulated under the form $u \in LB^\omega$, where L is some rational language of B^*, that we shall now compute in more details.

If the chronogram contains variables, we first identify the signal with an infinite word u on the alphabet C, as was explained before. The constraint in which the variables are not interpreted can be formulated under the form $u \in LB^\omega$, where L is some language rational of C^*, while the final constraint can be expressed under the form

$$u \in \bigcup_{\nu \text{ valuation}} \nu(L)B^\omega$$

There are in fact two types of constraints, the "static" constraints, which correspond to the case where L is a finite language, and the "dynamic" constraints, that correspond to the case where L can be an infinite language.

In the case of a static constraint, the language L is obtained as a finite concatenation of rational languages corresponding to static intervals. For instance, the following sequence of static intervals defines a constraint: "between time n_1 and n_2, the signal has a unique rising edge, between time n_2 and n_3, its value is a constant v and between time n_3 and n_4, its value is always 0". Note that in this case, the values of $n_2 - n_1$, $n_3 - n_2$ and $n_4 - n_3$ are the length of the static intervals.

In the case of a dynamic constraint, the language L is obtained as a finite concatenation of rational languages corresponding to dynamic intervals. For instance, the following sequence of dynamic intervals defines a constraint: "There exist instants n_2, n_3, n_4 such that between time n_1 and n_2, the signal has a unique rising edge, between time n_2 and n_3, its value is a constant v and between time n_3 and n_4, its value is always 0". The difference with the previous case is that the values of $n_2 - n_1$, $n_3 - n_2$ and $n_4 - n_3$ are not specified in the dynamic constraints, that is, can be chosen arbitrarily.

The languages associated with (static or dynamic) intervals are themselves obtained from the so-called *primitive* languages associated with the *primitive* symbols. This vocable concerns the elements of the set $V \cup \bar{V} \cup \{\mathbf{i}, \mathbf{0}, \mathbf{1}, \mathbf{f}, \mathbf{r}, \mathbf{s}, \mathbf{e}\}$ that is, all symbols of variables (possibly overlined) and the symbols associated with the graphic primitives of the Chronogram Language. First recall the intuitive meaning of these primitives.

i (Irrelevant) The value of the signal is not specified and can be either 0 or 1.
1 The signal is stable and its value is 1.
0 The signal is stable and its value is 0.
f (Falling) The signal owns one and only one falling edge (but may have 0, 1 or 2 rising edges).
r (Rising) The signal owns one and only one rising edge.
s (Stable) The signal is stable but its value is unknown.
e (Edge) The value of the signal changes once and only once.

This leads to the following table of the primitive languages associated with the graphic primitives:

$$L(\mathbf{i}) = \{0,1\}^+ \quad L(\mathbf{1}) = 1^+ \quad L(\mathbf{f}) = 0^*1^+0^+1^* \quad L(\mathbf{0}) = 0^+$$
$$L(\mathbf{r}) = 1^*0^+1^+0^* \quad L(\mathbf{s}) = 0^+ \cup 1^+ \quad L(\mathbf{e}) = 0^+1^+ \cup 1^+0^+$$

On the other hand, the primitive language associated with each variable v is

$$L(v) = v^+ \quad \text{and} \quad L(\bar{v}) = \bar{v}^+$$

We can now formally define the notion of interval. A *static interval* is a pair $I = (\ell, t)$ where ℓ is a positive integer and t is a primitive symbol. Intuitively, the integer ℓ represents the length of the interval on which the condition defined by t will be considered. For example, if $\ell = 5$ and $t = e$, the value of the signal will change once and only once in the interval $[0,5[$. The language associated with I is the subset of C^* defined by

$$L(I) = L(\ell, t) = L(t) \cap C^\ell$$

For example, if $\ell = 5$ and $t = e$, then $L(I) = (0^+1^+ \cup 1^+0^+) \cap C^5 = \{01111, 00111, 00011, 00001, 10000, 11000, 11100, 11110\}$ A *dynamic interval* is simply a primitive symbol and thus the corresponding language is already defined.

A *static* (resp. *dynamic*) row is a sequence of static (resp. dynamic) intervals. The language associated with a row $(I_1, I_2, \ldots, I_s)$ is defined by

$$L(I_1, I_2, \ldots, I_s) = L(I_1)L(I_2)\cdots L(I_s)$$

For instance, the language of C^+ associated with the row represented below is

$$11v\{0,1\}\bar{v}\bar{v}\{01111, 00111, 00011, 00001\}$$

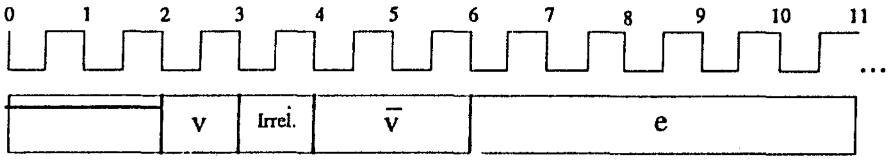


Figure 5.1. A static row

Here is another example, for a dynamic row. If $I_1 = v$, $I_2 = e$ and $I_3 = \bar{v}$, then

$$L(I) = v^+(0^+1^+ \cup 1^+0^+)\bar{v}^+$$

The languages associated with rows are described in the next proposition

Proposition 5.1. *The languages associated with a static row and their valuations are finite languages. The languages associated with a dynamic row and their valuations are languages of level* 3/2.

Finally, if L is the language associated with a (static or dynamic) row, the constraint defined by this row is the set

$$\bigcup_{\nu \text{ is a valuation}} \nu(L)B^\omega$$

In other words, in order to compute the constraint defined by a row, one first computes the language L associated with this row on the extended alphabet C and then one simply gives a value to the variables. For instance, for the row represented in figure 5.1, the constraint can be written $(111\{0,1\}00X \cup 110\{0,1\}11X)B^\omega$ where $X = \{01111, 00111, 00011, 00001\}$.

5.3. Constraints on several signals

We now define the language associated with a constraint on a set of k signals. We first introduce some auxiliary notation. Let A be an alphabet. For each integer k, A_k denotes the alphabet consisting of k-uple of letters of A, denoted as a column matrix. One can represent a word of length n on the alphabet A_k as a k-array of words of A^*. We denote by $\pi_A : A_k^* \rightarrow \underbrace{A^* \times A^* \times \cdots \times A^*}_{k \text{ times}}$ the function defined by

$$\pi_A\left(\begin{pmatrix} a_{1,1} \\ a_{1,2} \\ \vdots \\ a_{1,k} \end{pmatrix} \cdots \begin{pmatrix} a_{n,1} \\ a_{n,2} \\ \vdots \\ a_{n,k} \end{pmatrix}\right) = (a_{1,1} \cdots a_{n,1},\ a_{1,2} \cdots a_{n,2},\ \ldots,\ a_{1,k} \cdots a_{n,k})$$

This function π_A is in fact a monoid morphism of A_k^* into $A^* \times A^* \times \cdots \times A^*$: this simply means that it preserves the concatenation product. However, it is not an isomorphism (except if $k = 1$) because an element of $A^* \times A^* \times \cdots \times A^*$ may have components of different length. This leads to introduce the notation

$$D_k(A) = \{(u_1, u_2, \ldots, u_k) \in A^* \times A^* \cdots \times A^* \mid |u_1| = |u_2| = \ldots = |u_k|\}$$

to denote the set of k-tuples of words of the same length. Now, since π_A induces an isomorphism from A_k^* onto $D_k(A)$, one can identify the k-tuples of $D_k(A)$ with the words of A_k^*.

We now come back to signals. A *static multicolumn* is a pair $M = (w, R)$ where w is a positive integer and $R = (R_1, \ldots, R_k)$ is a k-uple of static rows. Intuitively, to each row corresponds a signal, but it is important to observe that two rows or more can represent the same physical signal. This technique allows to impose several distinct constraints on a given signal. By definition, the language associated with a static multicolumn (w, R) is

$$L(w,R) = C_k^w \cap \pi_C^{-1}(L(R_1) \times L(R_2) \times \cdots \times L(R_k) \cap D_k(C))$$

In other words, the k-tuples $(u_1, u_2, \ldots, u_k)$ such that $u_1 \in L(R_1)$, $u_2 \in L(R_2)$, $\ldots$, $u_k \in L(R_k)$ and $|u_1| = |u_2| = \ldots = |u_k| = w$ are selected and identified with words of C_k^*.
A *dynamic multicolumn* is a triple $M = (n, m, R)$ where n is a integer, m is either an integer or the symbol $*$ and $R = (R_1, \ldots, R_k)$ is a k-tuple of dynamic rows. The language associated with a dynamic multicolumn is by definition

$$L(n,m,R) = C_k^n \left(C_k^{[0,m]} \cap \pi_C^{-1}(L(R_1) \times L(R_2) \times \cdots \times L(R_k) \cap D_k(C))\right)$$
$$L(n,*,R) = C_k^n \left(C_k^* \cap \pi_C^{-1}(L(R_1) \times L(R_2) \times \cdots \times L(R_k) \cap D_k(C))\right) l$$

The difference between the types of multicolumns is that, in a dynamic multicolumn, there may be no upper bound on the common length of the u_i's.
One can show for the multicolumns a result similar to the one obtained for rows

Proposition 5.2. *The languages associated with a static multicolumn and their valuations are finite languages. The languages associated with a dynamic multicolumn and their valuations are languages of level* $3/2$.

5.4. Timing diagrams and properties, constraints

A *timing diagram* (TD) is a sequence of multicolumns. A *property* is a pair $P = (M, N)$ of timing diagrams : $M = (M_1, M_2, \ldots M_r)$ is the hypothesis and $N = (N_1, N_2, \ldots N_r)$ is the conclusion. The property is satisfied if every k-tuple of signals that satisfies the hypothesis satisfies the conclusion, too.

In the language formalism, this can be translated as follows: an infinite word w on the alphabet B_k satisfies a property $P = (M, N)$ if, for each suffix s of w, there exists a valuation ν such that for each factorization $u_1 u_2 \cdots u_r u_{r+1}$ of s, where $u_1 \in L_\nu(M_1)$, $u_2 \in L_\nu(M_2)$, $\ldots$, $u_r \in L_\nu(M_r)$, $u_{r+1} \in B_k^\omega$, there exists a factorisation $u_1' u_2' \cdots u_r' u_{r+1}'$ of s, where $u_1' \in L_\nu(M_1) \cap L_\nu(N_1)$, $u_2' \in L_\nu(M_2) \cap L_\nu(N_2)$, $\ldots$, $u_r' \in L_\nu(M_r) \cap L_\nu(N_r)$, $u_{r+1}' \in B_k^\omega$. This can be reformulated as follows.

Theorem 5.3. *An infinite word w on the alphabet B_k satisfies P if and only if none of its suffixes belong to the set*

$$K(P) = \bigcap_{\nu \text{ valuation}} L_\nu(M_1) L_\nu(M_2) \cdots L_\nu(M_r) B_k^\omega$$
$$\setminus \Big((L_\nu(M_1) \cap L_\nu(N_1)) (L_\nu(M_2) \cap L_\nu(N_2)) \cdots (L_\nu(M_r) \cap L_\nu(N_r)) \Big) B_k^\omega$$

Corollary 5.4. *An infinite word w on the alphabet B_k satisfies a property P if and only if it belongs to the set $L(P) = B_k^{\omega} \setminus B_k^* K(P)$.*

We arrive to our main result.

Theorem 5.5. *For every property P, the set $L(P)$ is a star-free set of level* 3.

We call a *constraint* a finite sequence of properties. The set of words defined by a constraint $(P_1, P_2, \ldots, P_n)$ is the language $L(P_1) \cap L(P_2) \cap \ldots \cap L(P_n)$. Now, the languages of level 3 are closed under intersection and thus Theorem 5.5 implies the following result.

Corollary 5.6. *The set of infinite words defined by a constraint is a star-free language of level* 3.

6. Conclusion

We have presented a new formal language for the specification of temporal properties of Discrete Event Dynamic Systems. This language, called the Chronogram Language, is based on a well-known graphic metaphor: waveforms. It allows specifying some complex temporal properties in a more convenient way than textual temporal logics (CTL, CTL* ...). Although we do not consider this language as a universal one, we think that its graphical approach might be appreciated by designers. In fact, we view the chronogram language as a basic part of a future Computer Aid Design (CAD) environment including validation tools.

The Chronogram Language is graphic and fully declarative. In this paper, we defined rigorously its semantics by using automata theory. The main result of this work is that it is possible to associate a finite automaton with any chronogram. In fact, as shown in this paper, chronograms correspond to a rather small subclass of the class of ω-rational sets.

We are now studying new developments: an extension of the Chronogram Language allowing designers to specify properties without any reference to some clock or including chronometric aspects (having physical time delays) and a consistency checking tool for sets of chronograms. We are also working on the improvement of the compilation algorithm since it is crucial to compile chronograms into as small as possible automata. Currently, compilers generate VHDL code and Signal code. New output languages will also be available in the future.

7. References

[1] C. Antoine, B. Le Goff, Timing diagrams for writing and checking logical and behavioral properties of integrated systems, P. Prineto and P. Camurati ed., *CHARME 91, Correct Hardware Design Methodologies*, Elsevier, Turin, Italy, 441–453, 1991.

[2] E. A. Ashcroft and W. W. Wadge, Lucid - a formal system for writing and proving programs, *SIAM J. Comp.* 5, 336–354, 1976.

[3] P. Caspi and N. Halbwachs, A functional model for describing and reasoning about time behaviour of computing systems. *Acta Informatica* **22**, 595–627, 1986.

[4] L. J. M. Claesen, editor, *Formal VLSI Correctness Verification*, IFIP, North-Holland, Amsterdam, November 1990.

[5] E. M. Clarke and I. A. Draghicescu, Expressibility results for linear-time and branching-time logics, In J. W. de Bakker, W. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in logics and Models for Concurrency*, 428–437, Springer-Verlag, June 1988.

[6] S. Eilenberg, *Automata, languages and machines*, Vol. A, Academic Press, New York, 1974, Vol. B, Academic Press, New York, 1976.

[7] E. A. Emerson, *Temporal and Modal Logic*, Chapter 16 in Handbook of Theoretical Computer Science (Van Leeuwen, J. ed.), Vol B : Formal Models and Semantics, Elsevier (1990).

[8] T. Hafer and W. Thomas, Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree, in T. Ottmann, editor, *ICALP'87*, Lecture Notes in Computer Science 269–279, Springer-Verlag, 1987.

[9] B. Le Goff, A. Benveniste, C. Figueira, and P. Le Guernic, CAD environment for real-time control system, in *American Control Conference ACC'89*, American Automatic Control Council, IEEE, Pittsburgh, Pennsylvania, June 1989.

[10] R. Lipsett, C.F. Schaeffer, Cary Ussery, *VHDL: Hardware Description and Design*, Kluwer Academic Publisher, Boston, Mass., 1990.

[11] J. Madre and J. Billon, Proving circuit correctness using formal comparison between expected and extracted behavior, in *Design Automation Conference, DAC'88*, 1988.

[12] J. S. Ostroff, A logic for real-time discrete event processes. *IEEE Control Systems, Magazine* **10**, 95–102, June 1990.

[13] J. S. Ostroff and W. M. Wonham, A framework for real-time discrete event control. *IEEE Transactions on Automatic Control* **35**, 386–397, April 1990.

[14] D. Perrin, *An introduction to automata on infinite words*, in Automata on infinite words (Nivat, M. ed.), Lecture Notes in Computer Science **192**, Springer (1984).

[15] D. Perrin and J.E. Pin, First order logic and star-free sets, *J. Comput. System Sci.* **32**, 1986, 393–406.

[16] D. Perrin, *Automata*, Chapter 1 in Handbook of Theoretical Computer Science (Van Leeuwen, J. ed.), Vol B : Formal Models and Semantics, Elsevier (1990).

[17] J.-E. Pin, *Variétés de languages formels*, Masson, Paris, 1984; English translation: *Varieties of formal languages*, Plenum, New York, 1986.

[18] W. Thomas, 1982, Classifying regular events in symbolic logic, *J. Comput. Syst. Sci* **25**, 360–375.

[19] W. Thomas, *Automata on infinite objects*, Chapter 4 in Handbook of Theoretical Computer Science (Van Leeuwen, J. ed.), Vol B : Formal Models and Semantics, Elsevier (1990).