Finite State Evaluation of Logical Formulas : Jevons' Approach (1870) and Contemporary Description

Paul Amblard*

Abstract

In this paper, we describe a formal language for a class of logical expressions. We then present a Finite State Machine for recognition and evaluation of this language. The main interest of the language is its historical characteristic. This language invented by the British scholar W. Stanley JEVONS in 1865 is probably the earliest language in which expressions were evaluated by a Finite State Machine. The two outstanding contributions were the use of machinery to evaluate formulas and the evaluation of formulas with variables by several parallel evaluations with constants. The contribution of this paper is to present this ancient evaluation process in a contemporary framework, i.e. formal languages and finite state automata. The design of an evaluator is given in great detail.

Introduction and Related Works

The history of calculating machines is well known. Pascal and Babbage built machines that are considered as the mechanical ancestors of today's computers. But computers do not only compute numbers, they can also perform symbolic evaluations. At a certain level of abstraction, we may consider mechanisms based on logical choices `if ... then ...`, or `if ... then ... else ...` as the necessary complements of strictly arithmetic operations.

The first mechanical machine making these choices to " perform the logical inference " was designed by William Stanley JEVONS in 1865 and published in 1870 ([14]). He had been a student of De Morgan. He was at that time becoming a professor of Logic (and of Political Economy) at the Owens College of Manchester. Figure 1 shows Jevons' activity time w.r.t other well-known British logicians.

His work has often been presented in the same terms as in the original paper: logical evaluation [1, 5, 6, 16]. Burris' paper [5] gives interesting details about Jevons' logic and about his machine. But none of these papers establishes relations

*TIMA-CMP Lab. University of Grenoble, 46 Av. Felix Viallet, 38031 Grenoble CEDEX, France, E-mail: `Paul.Amblard@imag.fr`

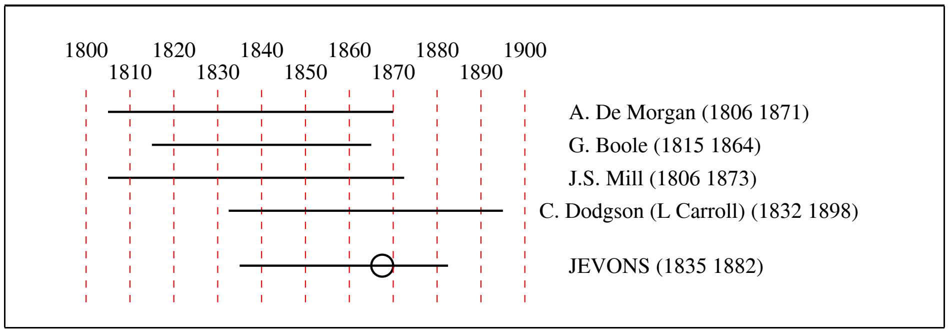

Figure 1: British logicians period. Jevons' time devoted to his machine is circled.

between Jevons' work and automata formalization. The only relations between Jevons' work and automata appear in Shepherdson's paper [19]. Unfortunately, after saying *And the key of the success of the whole endeavour was the discovery of a 'context free' algorithm which allowed the input proposition to be processed from left to right one symbol at a time*, the author did not explore in detail this 'context free algorithm'. He could certainly have discovered that it is in fact simpler than context free: no stack is used in this mechanical machine. However the paper is of great interest to us: Shepherdson describes the relations between the machine and the theory of Jevons, and he gives the drawings of the machine. A hardware implementation of this automaton by a V.L.S.I circuit had been studied in [2]. The present paper extends the presentation of [3].

In this paper we propose a description of Jevons' work in the framework of formal languages and automata. We shall see that Jevons can be considered as the inventor of Finite State transducers and evaluators. He also invented an ancestor in parallelism: input data are distributed (with modifications) to different "processors" running the same evaluation process, some kind of Single Instruction Multiple Data.

The paper is organized as follows: the first part describes a set of logical expressions. They constitute Jevons' formal language. The process of recognition is also described. The evaluation of this language is described in the second part. It is based on the Finite State paradigm. This part presents our technique and the technique used by Jevons himself to evaluate the formulas. It will then be possible in the third part to show that Jevons' evaluation, completed by syntactic analysis of formulas, meets our expectations. We shall give some details about the method used to obtain evaluation, based on composition of automata.

○ | In the paper some quotations from Jevons' presentation [14] will appear in this form. The number is the reference of the paragraph in the original text. ○

1 Jevons' Language

1.1 Pseudo-natural Language

Jevons dealt with logical formulas organized as

iron is metal AND *metal is not_wood*

The goal being obviously to deduce that

iron is not_wood

If we maintain Jevons' terminology, in the sentence *iron is metal* *iron* is a "subject", *is* is the "copula" and *metal* is an "attribute". The copula is obviously an implication.

He also allowed disjunctions of conjunctions both in the subject part and in the attribute part. This conjunction is denoted by *and*. Disjunction is denoted by *or*.

Jevons' language contained conjunctions of sentences. This conjunction is denoted by AND.

He could then write formulas like

iron and heavy is metal AND *heavy or metal is not_wood* AND *wood is not_metal or not_iron*

1.2 Formal Language Implemented by Jevons

In his formal language, Jevons used four variables A, B, C and D, and their respective complements a, b, c and d instead of natural language names (iron, metal,..) so the previous sentences become

A *is* B AND B *is* d

Obviously, the conclusion remains :

A *is* d

Jevons also made explicit the distinction between a variable appearing in an attribute part or in a subject part. Conjunctions of variables were denoted by simple concatenation, where

$$\mathrm{A}_s\mathrm{D}_s\mathrm{b}_s$$

simply denotes "A `and` D `and` not B" when this appears in a subject.

Similarly $\mathrm{A}_a\mathrm{D}_a\mathrm{b}_a$ appears in an attribute.

The disjunction was the inclusive OR. Let us remember that Boole used at this time the exclusive OR and that he and Jevons exchanged arguments about this choice. The generalization of inclusive OR is also a contribution from Jevons. The disjunction had also the distinction between subject and attribute giving $+_s$ and $+_a$.

The conjunction between sentences was an AND and was written as a "Full-Stop". This AND is syntactically different from the *and* between variables.

The language of correct expressions is described by Jevons but he did not give any formal description of it. Formal grammars were only invented 80 years later. Similarly automata were not already known with the contemporary meaning. The word already existed in Homer's Iliad (ch. 5, v. 749 and ch. 2, v. 408) but the reality is not the same. It refers to things (The gates of heavens) or people (Menelas) moving by themselves.

The photo of the keyboard on Jevons' machine is available from the website of the Museum of the History of Science in Oxford. Due to its aspect, many descriptions present it as the "Logical Piano" (`www.mhs.ox.ac.uk/images/index.htm` then search for Jevons).

36
The key board of the instrument is shown in fig. [..], where are seen two sets of term or letter keys, marked A, *a*, B, *b*, C, *c*, D, *d*, separated by a key marked COPULA–IS. The letter keys on the left belong to the subject of a proposition, those on the right to the predicate, and on either side just beyond the letter keys is a *Conjunction* key, appropriated to the disjunctive conjunction *or*, according as it occurs in the subject or predicate. The last key on the right hand is marked FULL STOP, and is to be pressed at the end of each proposition, where the full stop is properly placed. On the extreme left, lastly, is a key marked FINIS, which is used to terminate one problem and prepare the machine for a new one.

Example and transcription of Jevons' language in this paper

"$\mathrm{A}_s\mathrm{D}_s$ OR$_s$ $\mathrm{a}_s\mathrm{C}_s$ is $\mathrm{c}_a\mathrm{B}_a$ Full-Stop B_s is D_a OR$_a$ $\mathrm{A}_a\mathrm{c}_a$ Full-Stop" represent the formula nowadays written in standard logic as $(\mathrm{A}\wedge\mathrm{D} \vee \mathrm{a}\wedge\mathrm{C} \Rightarrow \mathrm{c}\wedge\mathrm{B}) \wedge (\mathrm{B} \Rightarrow \mathrm{D} \vee \mathrm{A}\wedge\mathrm{c})$ (We could also write $\neg$ A instead of a). In this paper we shall use the following form : $A_sD_s \ +_s \ a_sC_s \ \Rightarrow \ c_aB_a \bullet B_s \ \Rightarrow \ D_a \ +_a \ A_ac_a \bullet$.

We take the conventions

A_s, a_s, B_s, b_s, C_s, c_s, D_s, d_s are the variables in a subject part,
A_a, a_a, B_a, b_a, C_a, c_a, D_a, d_a are the variables in an attribute part,
$+_s$ and $+_a$ are the disjunctions, in subject and attribute part,
$\Rightarrow$ is the IMPLIES named "is" by Jevons,
$\bullet$ is the AND between sentences named "Full-stop" by Jevons.

Jevons added a key *Finis* which was simply a "reset" key. We do not use it because it simply forces the machine into the initial state.

Jevons' language can then be described by the grammar of figure 2. The vocabulary is V_T.

$$\mathrm{V}_T = \{A_s,\, a_s,\, B_s,\, b_s,\, C_s,\, c_s,\, D_s,\, d_s,\, +_s,\, \Rightarrow,\, A_a,\, a_a,\, B_a,\, b_a,\, C_a,\, c_a,\, D_a,\, d_a,\, +_a,\, \bullet\,\}$$

The grammar is described by noting that a problem is a sentence or a sentence followed by a problem, a sentence is a subject followed by an attribute, and so on. In the grammar, we use Var_s (resp. Var_a) for any variable in a subject (resp. attribute).

Problem	$\rightarrow$	Sentence	/ Sentence Problem
Sentence	$\rightarrow$		Subject $\Rightarrow$ Attribute $\bullet$
Subject	$\rightarrow$	Product$_s$	/ Product$_s$ $+_s$ Subject
Attribute	$\rightarrow$	Product$_a$	/ Product$_a$ $+_a$ Attribute
Product$_s$	$\rightarrow$	Var$_s$	/ Var$_s$ Product$_s$
Product$_a$	$\rightarrow$	Var$_a$	/ Var$_a$ Product$_a$

Figure 2: Grammar of Jevons' language

Another form of grammar is as follows :

Intermediate vocabulary is $V_N = \{$ J, K, L, M, N $\}$. The axiom is J. The rules are :

J $\rightarrow$ Var$_s$ K
K $\rightarrow$ Var$_s$ K / $+_s$ J / $\Rightarrow$ L
L $\rightarrow$ Var$_a$ M
M $\rightarrow$ Var$_a$ M / $+_a$ L / $\bullet$ N
N $\rightarrow$ Var$_s$ K / ϵ

Let us note, as a comment, that a variable can be repeated any number of times in a product without changing the meaning. We could then think about an asynchronous automaton as in ([13]) but this property is not true for other symbols. We should have to admit "strange" expressions such as $1_s \Rightarrow \Rightarrow 0_a \bullet$.

Another comment is about redundancy between the indication subject-attribute and the correct alternation of the separators $\bullet$ and $\Rightarrow$.

The finite state recognizer of the language is represented by Automaton `SA` in figure 3.

2 Evaluation of Formulas

The main contribution of Jevons concerns evaluation of the aforesaid logical formulas. His method was obviously not explicitely based on Finite State Transducers, but, as we shall see, all the ideas were already present. His method was based on two levels: the first one consists of the evaluation of a formula containing variables by implementing several evaluations of formulas containing only constants (true 1, false 0). The second level is indeed a Finite State Evaluation process. The combination of the two levels could be described as a Single Instruction Multiple Data machine.

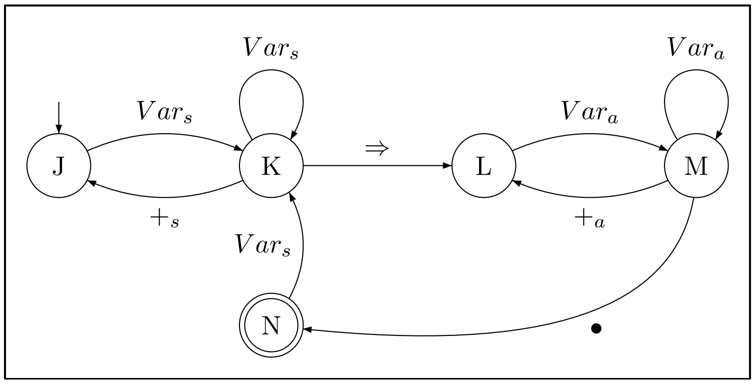

Figure 3: `SA` : Syntax Analyser. Recognition of Jevons' formulas. J is initial state, N is final state. The "dead" state is hidden. All transitions not described lead to this hidden state.

2.1 From Variables to Constants

Jevons considered evaluation of formulas with 4 logical variables A, B, C, D. To perform this evaluation, he considered 16 situations, corresponding to the 16 lines of a (nowadays) classic truth table. Truth tables were already known in 1870, in some forms, mainly presented by Leibniz. Jevons used truth tables under the name of *logical abecedarium*.

> 20
> Problems involving four distinct terms would similarly require a series of sixteen conceivable combinations, and if five or six terms enter, there will be thirty-two or sixty-four of such combinations. These series of combinations appear to hold a position in logical science at least as important as that of the multiplication table in arithmetic or the coefficients of the binomial theorem in the higher parts of mathematics. I propose to call any such complete series of combinations a *Logical Abecedarium*...

To evaluate a formula with N variables, Jevons simply evaluates 2^N formulas with constants. Each individual evaluator is labelled by a name representing a line in the truth table. Line ABCD represents the line where both A, B, C, D are true, line AbCd represents the line where A and C are true and B and D are false, and so on.

Jevons designed his machine with such a mechanism that evaluation of

$$A_s D_s \ +_s \ a_s C_s \ \Rightarrow \ c_a B_a \ \bullet \ B_s \ \Rightarrow \ D_a \ +_a \ A_a c_a \ \bullet$$

is implemented by evaluating

$1_s 1_s +_s 0_s 1_s \Rightarrow 0_a 1_a \bullet 1_s \Rightarrow 1_a +_a 1_a 0_a \bullet$ on line ABCD
$1_s 0_s +_s 0_s 1_s \Rightarrow 0_a 1_a \bullet 1_s \Rightarrow 0_a +_a 1_a 0_a \bullet$ on line ABCd
$1_s 1_s +_s 0_s 0_s \Rightarrow 1_a 1_a \bullet 1_s \Rightarrow 1_a +_a 1_a 1_a \bullet$ on line ABcD

and so on.
The fact that these 16 evaluations could be done in parallel was more obvious at that time. Jevons was not disturbed by the sequential activities scheme introduced by modern computers under the Von Neumann paradigm.

○ The Logical Abacus was devised [..] and was constructed by placing the combinations of the *abecedarium* upon **separate moveable** slips of wood, which can then be easily classified, selected and arranged according to the conditions of the problem. ○

The mechanism moving the different slips of wood was slightly different for each line of the Abecedarium. So we see that the standard problem SAT, known to be NP-complete, was first solved by a system responding in constant time (in fact in time 0, the answer is given immediately at the end of the formula) but exponential in number of processors.

In Jevons' machine, however, the energy for activating the 16 evaluators was simply given by the user pressing the key of a keyboard. This energy limited the parallelism degree of the system.

2.2 Evaluation of Formulas with Constants

From a timed sequence of inputs (actions on mechanic keys) the Jevons' machine delivered, after each input, a result giving a temporary evaluation of the formula. This result contains 16 evaluations, on the 16 lines. The result of one basic evaluation is simply `true` or `false`. By giving these 16 results, the machine gives in a certain way the valuation which makes the formula `true`. At some pre-established instants, this evaluation is in adequation with the expected result. We may choose to consider results only after an AND (represented by a $\bullet$, separating sentences). We shall consider two techniques of evaluations: ours is based on an extension of the syntactic acceptor, then we shall give Jevons' proposal.

2.2.1 Our evaluation, based on syntactic recognition

We can give a value to any problem by the function Val. It gives a boolean result, based on the boolean values of the basic "Bo" atoms and the laws of boolean algebra. "Bo" stands for a boolean, 1 or 0. The description of Val is related to the grammar given in figure 2.

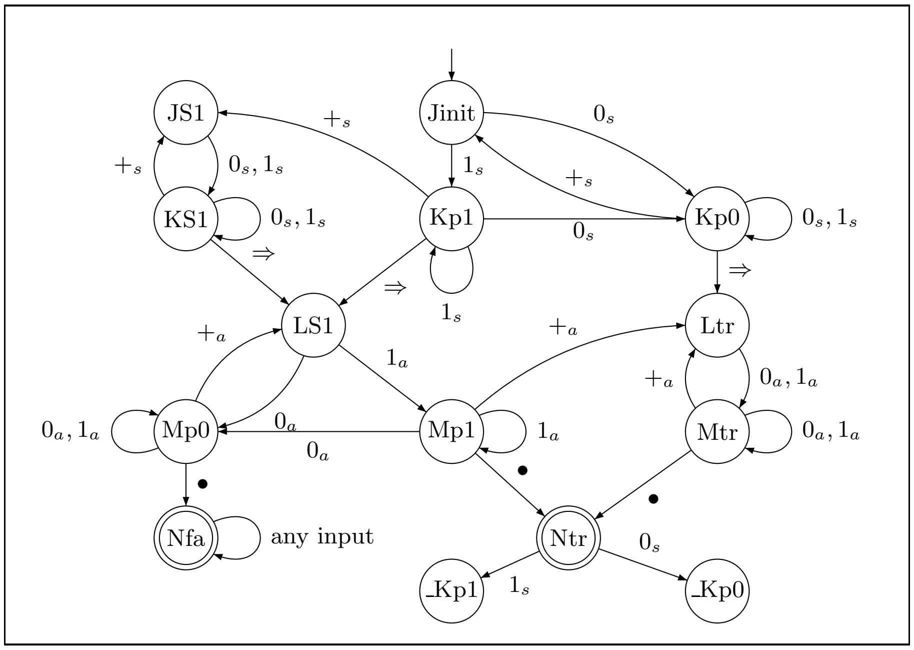

Figure 4: `SE` : Syntax-based Evaluation. States Kp1 and Kp0 are represented twice to make the figure easier to read. Ntr has the same successors as the initial state Jinit. In Ntr the expression evaluates to `true`, in Nfa, it evaluates to `false`

Val (Pr)	= Val (Sent) ;	Val (Pr)	= Val (Sent)	AND Val (Pr)
Val (Sent)	=		NOT Val (Sub)	OR Val (Attr)
Val (Sub)	= Val (Pro$_s$) ;	Val (Sub)	= Val (Pro$_s$)	OR Val (Sub)
Val (Attr)	= Val (Pro$_a$) ;	Val (Attr)	= Val (Pro$_a$)	OR Val (Attr)
Val (Pro$_s$)	= Val (Bo$_s$) ;	Val (Pro$_s$)	= Val (Bo$_s$)	AND Val (Pro$_s$)
Val (Pro$_a$)	= Val (Bo$_a$) ;	Val (Pro$_a$)	= Val (Bo$_a$)	AND Val (Pro$_a$)

We have extended the recognition automaton by considering the values of the interesting booleans evaluated in the machine. They are

– the current conjunction (or Product) of variables,
– the current disjunction (or Sum) of conjunctions, (and we must remember the subject Sum and the attribute Sum)
– these two sums give the value of the current sentence, by $x \Rightarrow y = \neg x \vee y$
– the current value of the conjunction (product) of sentences.

A further section (3.2) will describe the process to obtain this automaton. Let us summarize the correspondence between the states of the syntactic recognizer (figure 3) and the states of our evaluator (figure 4) :

- In state J, either the subject is already certainly `true` after a first true product in a sum (state JS1), or the subject is not already certainly `true` (state Jinit).
- In state K, if the subject was already `true`, it remains (state KS1). If the subject is not yet `true`, either the current product is `true` (state Kp1) or the current product is `false` (state Kp0).
- In state L, either the subject was `true` (state LS1) or the sentence is already certainly `true` (state Ltr). This occurs either when the subject is `false` or when the subject is `true` and the attribute is already certainly `true`.
- In state M, if the sentence is already certainly `true`, it remains (state Mtr). In the other case, (the subject was certainly `true` and the attribute is not already certainly `true`), either the current product of the attribute is `false` (state Mp0) or this product is `true` (state Mp1).
- In state N, just after a •, the sentence is evaluated and the product of all the sentences is generated. When it has been `false` once (state Nfa), it remains false. If all the previous sentences have been `true` (state Ntr), evaluation goes on.

2.2.2 Jevons' evaluation

The goal of the present paper is not to present the method proposed by Jevons and its relations to "inductive" logic. This is done in ([6], [16] and [19]). It can also be understood from the original text. The basics is principally that we are interested in a prefix of expression. This prefix is a previous sentence, terminated by a •, followed by a premise. Then any line in the truth table can be classified in one of four categories with respect to the given prefix. (Line excluded by the premise, Line included and consistent with the premise, Line inconsistent with the premise, Line inconsistent with the previous sentence). These four situations can be modeled by four states.

We may consider how Jevons himself would have described the four states of one automaton (part 39) and the transitions due to action of the key "Full stop" (part 41).

> 39
> It is now necessary to explain that each rod has four possible positions fully indicated in the figs. [–]. The **first** of these positions is the neutral or initial position []. The **second** position is that into which a rod is thrown by a subject key ; the **third** position lies in the opposite direction, and is that into which a rod is thrown by a predicate key. The **fourth** position lies one half inch beyond the third. The **four** positions evidently correspond to the four classes into which combinations were classified in the previous part of the paper []

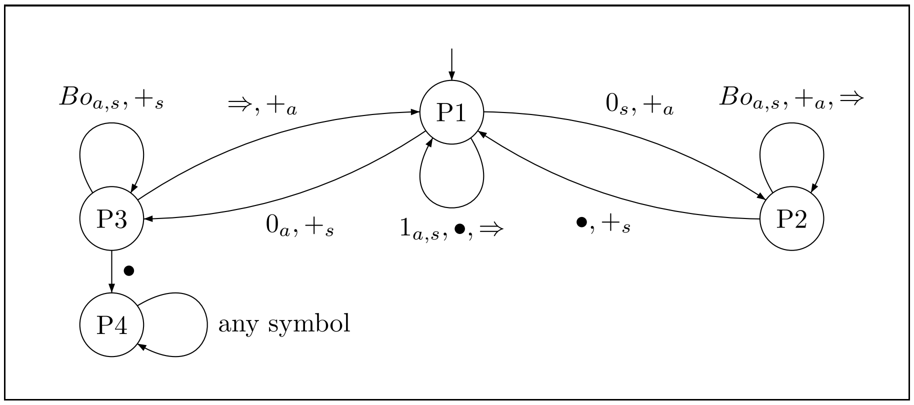

Figure 5: `JE` : Jevons' Evaluation : Reconstructed Jevons' automaton for evaluation. In state P1, the formula is `true`, in state P4, it is `false`. Bo is one of the booleans, 0 or 1.

His four positions are our four states P1, P2, P3 and P4 appearing in figure 5.

> 41
> The **full-stop** key being now pressed has a double effect. It acts [on the pins and rods of the machine] These pins we may distinguish as the α and β pins, the α pin being the uppermost. While a rod is in the **first** position the lever [] has **no effect** ; but if the rod be lowered $\frac{1}{2}$ inch into the **second** position, the lever will cause the rod to return to the **first** position by means of the α pin ; but if the rod be raised into the **third** position, the β pin will come into gear, and the rod will be pushed $\frac{1}{2}$ inch further into the **fourth** position.

Part 41 clearly describes a part of the transition function `succ` for the same input •, and the four states.

$$\texttt{succ}(\text{P1}, \bullet) = \text{P1} \; ; \; \texttt{succ}(\text{P2}, \bullet) = \text{P1} \; ; \; \texttt{succ}(\text{P3}, \bullet) = \text{P4} \; ;$$

From these different explanations, we could infer the 4 state machine given by figure 5. In P1, the sentence is `true`. In P4, the sentence is `false`.

It is possible to follow the evaluation according to Jevons' technique on two tables: in these tables the initial state is P1. After a given input (first line), the new state is given under this input (second line).

For a formula giving a `true` result :

0_s	=>	0_a	$+_a$	0_a1_a	•	0_s0_s	$+_s$	1_s0_s	=>	1_a0_a	•
2	2	2	2	2 2	1	2 2	1	1 2	2	2 2	1

and similarly for a `false` result :

0_s	$=>$	1_a	$+_a$	0_a0_a	$\bullet$	0_s1_s	$+_s$	1_s1_s	$=>$	0_a0_a	$\bullet$
2	2	2	2	2 2	1	2 2	1	1 1	1	3 3	4

3 Jevons' Evaluation Coupled with Syntactic Recognition

Jevons did not verify the syntax of the formula entered on the keyboard. He was probably the only user of the machine in his courses of Logic. There were probably no syntax errors in his inputs! We have tried to compose evaluation and recognition by computing product automata. We used two evaluators: Jevons' one, of course, and an evaluator obtained by composing more basic evaluators.

3.1 Equivalence between two Evaluations

There are different definitions of the product automaton. We take the one of ([12], pp 134-137). The product computes the `AND` of the two composed automata. When we deal with recognition, it corresponds to intersection of languages. Here the interpretation is different, but `AND` is possible because the evaluator delivers a boolean (the value of the formula) and the recognizer can also be described with such a boolean output. If we name `SA` the syntactic analyser of figure 3 and `JE` the Jevons' evaluator of figure 5 the product `SA` $\times$ `JE` gives `SE`, the automaton of figure 4. (The correspondence between states is given by the Cartesian product of states in figure 6).

This composition is an interesting result. We can consider it as a validation of our syntactical evaluation method.

To enter more deeply into Jevons' technique, the reader may draw surrounding shapes on figure 4.

- One shape labelled P3 around states JS1, KS1, Mp0;
- One shape labelled P1 around Jinit, Kp1, LS1, Mp1, Ntr;
- One shape labelled P2 around Kp0, Ltr and Mtr.

The next section will give details about the design of `SE`.

3.2 Evaluation by Composition of Basic Evaluators

We have tried without success to obtain Jevons' automaton `JE` by composition of more basic understandable automata. We only obtained a composition evaluating *correct* formulas in the same way as Jevons' method (`JE`). We obtained `CE` our composed evaluator (figure 12), `JE` and `CE` are not equivalent. The (wrong) expression $0_s \bullet$ does not give the same values in the two processes. If we compose them by the Syntax Analyser `SA`, the products `SA` $\times$ `JE` and `SA` $\times$ `CE` are equivalents. In both cases we obtain `SE`, the automaton of figure 4.

evaluation \ *syntax*	J	K	L	M	N
P1	Jinit	Kp1	LS1	Mp1	Ntr
P2		Kp0	Ltr	Mtr	
P3	JS1	KS1		Mp0	
P4	Nfa	Nfa	Nfa	Nfa	Nfa

Figure 6: Correspondence of states between the automaton of composite Syntax_Evaluation (fig. 4 `SE`) and the product of the Syntax Analyser (fig. 3 `SA`) by the Jevons' Evaluation automaton (fig. 5 `JE`)

Where does `CE` come from ? `CE` is the result of composing 5 automata. The organization of composition is given by figure 7. The composition has been computed with LUSTRE environment described in the next subsection. In the same way, the equivalences have been checked with this tool.

Automaton `PROD` (figure 8) is a Moore automaton, it receives all the inputs and delivers the product's value `P`.

Automaton `SUM` (figure 9) is a Mealy automaton, it receives symbols ($+_a$, $+_s$, $\Rightarrow$, $\bullet$) and the value `P` delivered by `PROD`. It delivers the sum of products value `S`.

Automata `SUBJ` (figure 10) and `ATTR` receive symbols ($\Rightarrow, \bullet$) and the value `S` of the sum of products. They deliver (respectively) the values `Su` and `At` of subject and attribute part. They are Mealy automata. `SUBJ` takes the value of S into account when $\Rightarrow$ occurs. In a symetric way, `ATTR` deals with `S` when $\bullet$ occurs.

Automaton `EXPR` (figure 11) is a Mealy automaton. It receives symbols ($\Rightarrow$, $\bullet$) and the values of `Su` and `At`. It delivers the global value `Ex` of Jevons' expression.

All these automata have two states as we could expect from boolean evaluators.

3.3 The Language Lustre and the Environment

The language LUSTRE has been designed in the '80s for real-time programming [10, 11]. The present description contains only some basic points useful to understand the composition made with the automata. The same approach is used for the environment. The use of LUSTRE in education is described in [4].

3.3.1 Boolean LUSTRE

Boolean LUSTRE has only one type : `boolean`. The boolean operations (`not`, `and`, `or` and `xor`) are defined in boolean LUSTRE. Two timed operators (`pre` and $->$) allow us to deal with unitary delay and initialization. The synchronous hypothesis is that the automata update their states at the same clock ticks.

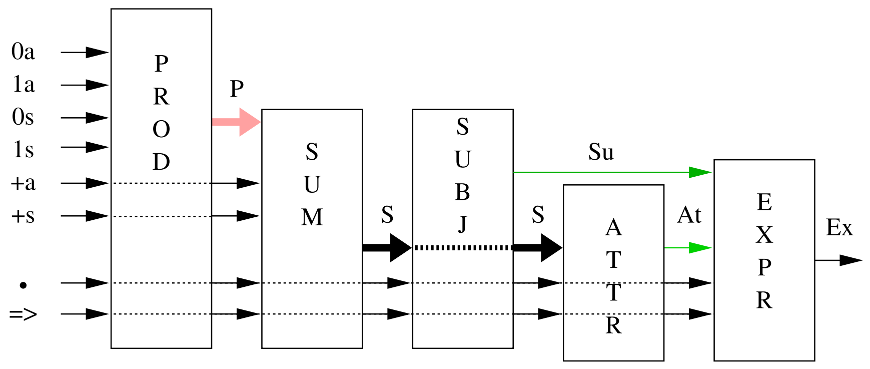

Figure 7: Composition of 5 basic automata to obtain an evaluator. It is inspired by the organization of a sequential circuit. Each input symbol is considered as a "wire", true or false at any instant. One and only one of these wires is true at any instant. These combinations represent the occurences of one symbol. P, S, Su and At are internal variables. The circuit would be a synchronous one, the clock being common.

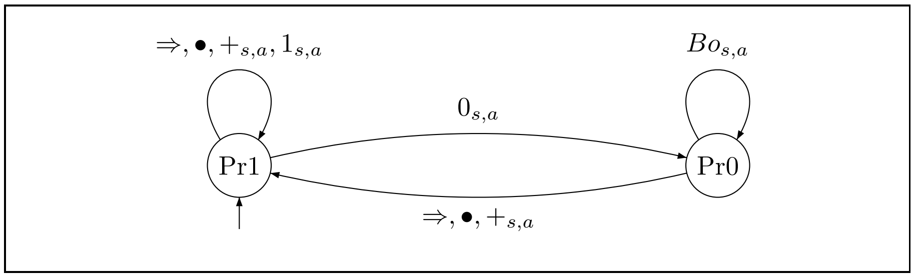
$\Rightarrow, \bullet, +_{s,a}, 1_{s,a}$
$Bo_{s,a}$
$0_{s,a}$
Pr1
Pr0
$\Rightarrow, \bullet, +_{s,a}$

Figure 8: `PROD` : Evaluation of products. In state Pr1, the product `P` is 1, in state Pr0, the value is 0. The product is reset at 1 when a separator ($\Rightarrow$, $\bullet$, $+$) occurs and this product becomes 0 only when a 0 boolean occurs.

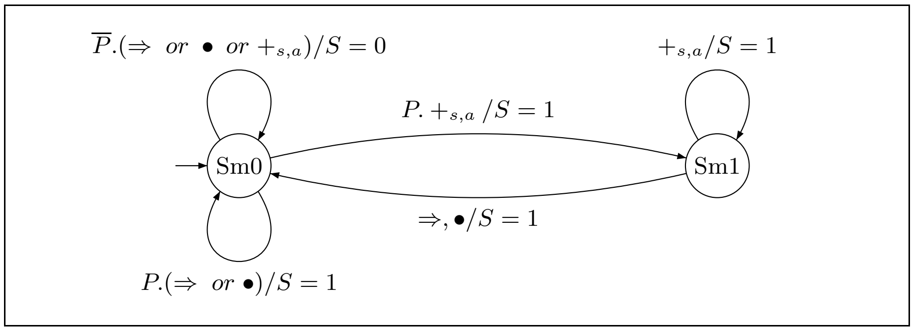
$\overline{P}.(\Rightarrow\ or\ \bullet\ or\ +_{s,a})/S=0$
$+_{s,a}/S=1$
$P.+_{s,a}\ /S=1$
Sm0
Sm1
$\Rightarrow,\bullet/S=1$
$P.(\Rightarrow\ or\ \bullet)/S=1$

Figure 9: `SUM` : Evaluation of sum `S`, from the values of the products. P stands for (Product is 1) and $\overline{P}$ for (Product is 0). Booleans are not taken into account, updating only occurs on separator occurences.

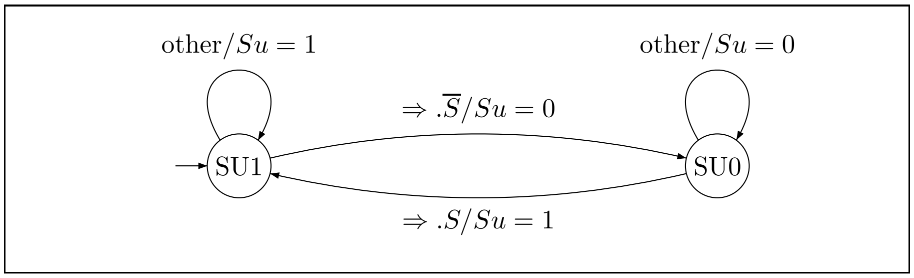

Figure 10: `SUBJ` : Evaluation of subjects `Su` from the value of the sum `S`. Updating occurs when $\Rightarrow$ occurs. Subject's value then receives the value of the sum `S`.

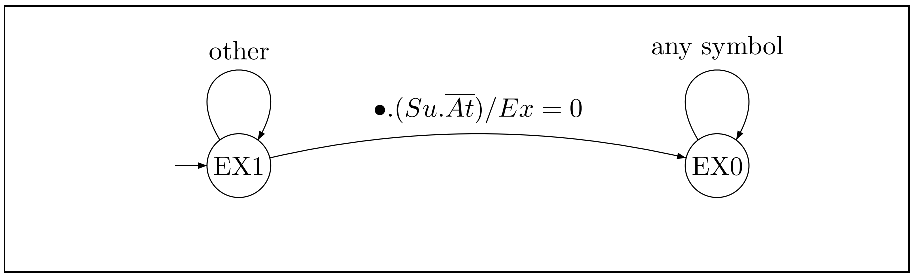

Figure 11: `EXPR` : Evaluation of a Jevons' expression `Ex` from Subject and Attribute's values. An expression remains `true` until occurence of a $\bullet$ when the current implication is `false`, i.e. the current subject Su is `true` and the current attribute At is `false`.

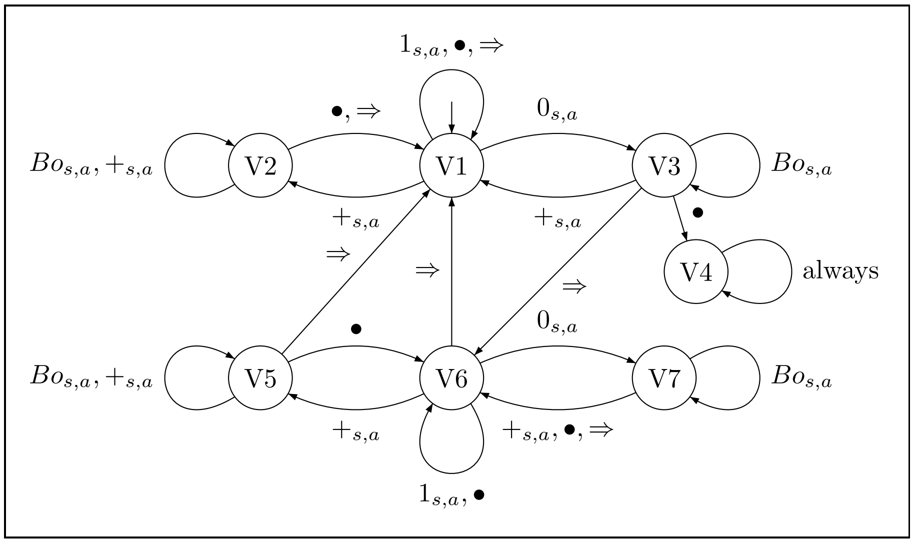

Figure 12: `CE` : Composed Evaluation. Evaluator obtained by combining `PROD, SUM, SUBJ, ATTR, EXPR`. In state V4, the expression is conclusively false. In states V1 and V6, it is true.

3.3.2 Basic descriptions of automata

Boolean LUSTRE makes it possible to describe finite automata in many different styles :

- The automaton can simply be described by a classical set of states, a description of the transition function and the description of the output function. This automaton may be deterministic or not, complete or not.
- We must introduce a comment about our mode of description of language recognizers. The only type being `boolean`, we cannot have a vocabulary based on characters. We solve this problem by introducing a set of boolean inputs such as {a, b, c, d}. We need to avoid the problem of 16 possible values of these four booleans. We use `assert` constructs to constrain one and only one amongst {a, b, c, d} to be true at any time.
- The general use of automata in formal languages studies distinguishes acceptor states and not_acceptor ones. This is obviously equivalent to having a boolean output defined in {0, 1} for each state. If we deal with more general automata, with not-simply boolean outputs, we may use the same approach. We then declare as many booleans as useful outputs.
- If the global automaton is not known, we can give *properties* of the automaton. This method is powerful but no systematic rules can be given. We may

experiment if the properties are adequate or not. Example of property is *For any transition due to input symbol X, a state and its successor never give the same output.* Experimentations could be simulation or formal proofs.

- When we deal with the *synchronous* sequential digital circuits, the description can easily be given in boolean LUSTRE. Logical gates are described by the operators. If we want to be close to the implementation we may describe `nand` or `nor` gates. Flip-flops are described by the timed operators.
- A systematic method of description of a regular grammar exists in boolean LUSTRE but there are restrictions on the form of the grammar: if A and B are non-terminal symbols and if x is a terminal, rules must be expressed in one of the two forms where we recognize initialization and unitary delay: $A \rightarrow \epsilon$ or $A \rightarrow B \, . \, x$
- A translator exists from a language allowing to describe regular expressions. This tool is described in [18].

3.3.3 Combinations of automata

Boolean LUSTRE allows to combine objects as it is the case in general LUSTRE. Different combinations of objects are possible :

- A very common case is that two automata A1, A2 are defined by LUSTRE nodes N1, N2 with the same inputs (*inp*). Both boolean LUSTRE nodes deliver one boolean output. A boolean operator `OP` allows us to define a new node as
  N3 (*inp*) = N1 (*inp*) `OP` N2 (*inp*). It creates a composed automaton A3. The language L3 recognized by A3 is a function of languages L1 and L2 recognized by A1 and A2. It also corresponds to the introduction of a logical gate on the two output signals of the two circuits.

  Correspondence between gates and language operations are obvious: `not` gate gives the complementary language, `and` gate gives the intersection of languages. ([12], p 135).
  `not xor` gate is particular. If two automata have always the same output, the composed automaton delivers always the output `true`. This corresponds to computing the equivalence of two automata. It is used to prove equivalence between two descriptions of automata that are assumed to be equivalent. (Similarly a $\Rightarrow$ operator is used to test inclusion of languages.)
- It is also possible to do cross-coupling of two nodes: some inputs of a node are outputs of the other one or vice-versa. It is very common in circuits. We must not include combinational loops.
- Any serial or parallel composition of automata may be described. An example appears in figure 7.

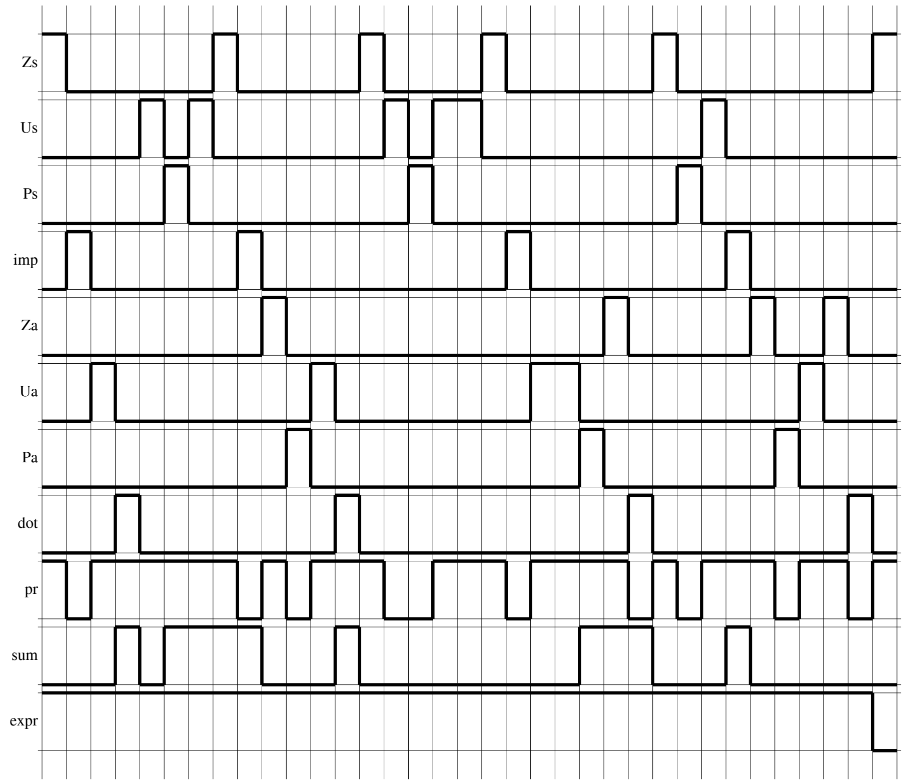

Figure 13: Simulation in Lustre. Zs represents 0_s, imp represents $\Rightarrow$, pr is the current value of the product. The input sequence begins with $0_s \;\Rightarrow\; 1_a \bullet$. The time slices are represented on the bottom line.

3.3.4 Environment and uses

The environment is available ([22]) under Solaris or Linux. Three tools are used in our work.

- The LUSTRE simulator allows to visualize the behaviour of the given object. The results are given in textual form or in timing diagrams form. This is particularly standard in digital circuits simulation. Figure 13 shows a simulation result of an evaluation. One character (represented by a boolean, true when occuring) is represented by one line of the oscilloscope.

- The LUSTRE combiner_minimizer computes the finite state machine described in input. The result of this compilation is a full definition: (list of states, list of all transitions). The result automaton is complete, deterministic and minimal. Obviously if we described a complete deterministic minimal automaton as input, the compilation is only a state renaming ! It is particularly useful in composing automata. Obviously we must remain aware that the combinatorial explosion is possible.

- We use the Lesar tool in a particular case: for automata with only one boolean output, such as recognizers, Lesar computes if this output is always `true`. If it is not the case, Lesar gives a counterexample. This counterexample is very interesting when we test automata equivalence.

4 Conclusions

Obviously the contribution made by Jevons was an important step in the mechanization of Logic. The first machine devoted to artificial intelligence was his. The fact that syntactical aspects were not covered is easy to understand. But it is very pleasant to discover, by simple techniques, that his method could have been coupled with Finite State recognition. The present paper introduced the details about possibilities of such a composition with LUSTRE environment.

Part 46 of the original text opens a new problem: due to mechanical implementation, it was possible to press several keys simultaneously. Do we have to change automata theory to take such a feature into account?

46
When several of the letter keys on the subject side only or the predicate side only are pressed in succession, the effect is to select the combinations possessing all the letters marked on the keys. Thus if the keys A, B, C be pressed there will remain in the abecedarium only the combinations A B C D and A B C *d* ; and if the key D be now pressed, the latter combination will disappear, and A B C D will alone remain. The effect will be exactly the same whatever the order in which the keys are pressed, and if they be pressed **simultaneously** there will be no difference in the result.

Acknowledgements

The drawing of automata used Latex packages made by Paul Gastin. The contribution of Lustre developers was obviously necessary.

References

[1] S.G. Akl, *Professor Jevons and his Logical Machine*, The Australian Computer Bulletin, June 1981, pp 28-30.

[2] P. Amblard, *A VLSI Implementation of the Earliest Specialized Logical Computer : the Jevons' Machine*, 4th International Workshop Mixed Design of integrated circuits and systems, Poznan, Poland, Mixdes97, June 1997, pp 55-66.

[3] P. Amblard, *The Earliest Formal Language and its Associated Finite State Evaluation Automaton : Jevons' Machine*, 11th International conference Automata and Formal Languages, Dobogókő, Hungary, May 2005, pp 59-68.

[4] P. Amblard *Using Lustre in Practical Educational Activities : Digital Circuits Design, Formal Languages*, ETAPS Workshop : Synchronous Language Applications Programming, SLAP 05 Edinburgh, April 2005.

[5] G.H. Buck, S.M. Munka, *W. Stanley Jevons, Allan Marquand, and the Origin of Digital Computing*, IEEE Annals of the History of Computing, Vol 21, No 4, October-December 1999, pp 21-27.

[6] S.N. Burris, *Contributions of the Logicians*, part 1 From Richard Whately to William Stanley Jevons, on-line : `www.thoralf.uwaterloo.ca/`

[7] I. Casltes, *Vice President's note*, Newsletter of the Academy of the Social Sciences in Australia, Vol 17, No 3, 1998, pp 5-12; `www.assa.edu.au/publications/Dialogue/dial31998.pdf`

[8] M. Gardner, *Logic Machines*, Scientific American, March 1952, pp 68-73.

[9] M. Gardner, **Logic Machines and Diagrams**, McGraw-hill 1958, and The Harvester Press, Brighton, 1983

[10] N. Halbwachs, **Synchronous Programming of Reactive Systems**, Kluwer Academic Pub., 1993

[11] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, *The Synchronous Dataflow Programming Language Lustre*, Proceedings of the IEEE, September 1991, pp 1305-1320.

[12] J.E. Hopcroft, R. Motwani, J.D. Ullman, **Introduction to Automata Theory, Languages and Computation**, Addison Wesley, 2001.

[13] B. Imreh, *Some Remarks on Asynchronous Automata*, Conference DLT 2002, Lect. Notes in Comp. Science No 2450, pp 290-296.

[14] William Stanley Jevons, *On the Mechanical Performance of Inference*, Philosophical Transactions of the Royal Society, London, 1870, pp 497-518. Available on-line : `tima-cmp.imag.fr/~amblard/JEVONS/jevons_public.pdf`

[15] W. S. Jevons, **Papers and Correspondence**, (ed : R.D. Collison Black and Rosamond Kőnekamp), Vol 3, (correspondence 1863-1872), MacMillan Press, (London), 1977, pp 69-76.

[16] W. Mays, D. P. Henry, *Jevons and Logic*, Mind, Vol LXII, 1953, T. Nelson & sons, Edinburgh, pp 484-505.

[17] E. F. Moore, *Gedanken-experiments on Sequential Machines* in **Automata studies**, (Ed : C. Shannon, J. McCarthy) Princeton University Press, 1956, pp 129-153.

[18] P. Raymond, *Recognizing Regular Expressions by means of Dataflows Networks*, 23rd International Colloquium on Automata, Languages, and Programming, (ICALP'96) Paderborn, Germany, July 1996, LNCS 1099.

[19] J.C. Shepherdson, *W. S. Jevons: his Logical Machine and Work on Induction and Boolean Algebra*, Machine Intelligence 15, Oxford, July 1995, pp 489-505.

[20] **Dictionnary of National Biography**, vol XXIX, Smith, Elder and co, London, 1892, pp 374-378.

[21] **Sequential Machines, Selected papers**, Ed : E. F. Moore, Addison-Wesley, 1964.

[22] Web-site : `www-verimag.imag.fr/SYNCHRONE`