**1) Abstarct**

This report will describe the LUSTRE Programing which is a data flow synchronous language, designed for programming reactive system such as automatic control and monitoring systems as well as for describing hardware. The data flow aspect of LUSTRE makes it very close tousual description tools in these domains (block diagrams, networks of operators, dynamical sample-systems and its synchronous interpretation makes it well suited for handling time in programs. Moreover, this synchronous interpretation allows it to be compiled into an efficient sequential program. Finally, the LUSTRE formalism is very similar to temporal logics. This allows the language to be used for both writing programs and expressing program properties, which results in an original program verification methodology.

**2) Brief Introduction**

LUSTRE is based on the synchronous paradigm which is the behavior of a program is a sequence of reactions, each reaction consisting of reading the current inputs, computing the current outputs, and updating the internal state. So, a program typically implements an automation the states are the valuations of the memory, and each reaction corresponds to a transition of the automation. Such a transition may involve many computations, which, from the automation point of view, are considered input changes are only taken into account between two reactions).

Besides being synchronous, LUSTRE is also data-flow. The goal is to adhere to the common formalisms of control engineers, which are often data-flow synchronous formalisms, inherited from earlier analog technology: differential or finite-difference equations block diagrams, analog networks. Interpreted in a discrete world, these models can be formalized using the data-flow paradigm.

**3) Literature Review**

LUSTRE is a [synchronous](http://en.wikipedia.org/wiki/Synchronous_programming_language" \o "Synchronous programming language) [dataflow programming](http://en.wikipedia.org/wiki/Dataflow_programming) language for programming reactive systems. It began as a research project in the early 1980s. A formal presentation of the language can be found in the 1991 Proceedings of the IEEE. In 1993 it progressed to practical, industrial use in a commercial product as the core language of the industrial environment [SCADE](http://en.wikipedia.org/w/index.php?title=SCADE&action=edit&redlink=1" \o "SCADE (page does not exist)), developed by [Esterel Technologies](http://en.wikipedia.org/wiki/Esterel_Technologies" \o "Esterel Technologies). It is now used for critical control software in [aircraft](http://en.wikipedia.org/wiki/Airbus" \o "Airbus), [helicopters](http://en.wikipedia.org/wiki/Eurocopter_Group), and [nuclear power plants](http://en.wikipedia.org/wiki/Nuclear_power_plants" \o "Nuclear power plants).

LUSTRE is a programming language founded on these remarks. A program is a system of equations defining variables, which are functions from time to their domain of values. Since we are concerned with discrete systems, time is projected onto the set of naturals, making variables infinite sequences of values. Furthermore, a program may be viewed as an operator net, aa is standard for data-flow languages, with a further assumption called synchrony, which states that operators respond instantaneously to their input

Our equational point of view may be summarized by the two following principles:

Substitution principle an equation X=E specifies a full synonymy between the variable X and the expression E. Thus, in every context, the identifier X may be replaced by the expression E, and conversely. This property is very useful in program transformation.

Definition principle Let X=E be the equation defining the variable X. Then the behavior of X must be completely specified by this equation and the behavior of variables appearing in the expression E. A program defines a function from its input (sequences) to its output (sequences). From the assumption of synchrony, all functions expressible in the language must satisfy the following properties:

1. Causality The output at any instant t may only depend upon input received before or at t. Notice that in this sense, LUCID [l], a close parent of LUSTRE, allows the definition of unusual programs.
2. Bounded memory. There must exist a finite bound such that, at each instant, the number of past input values that are necessary to produce the present and future output values remains smaller than that bound.

**4) How the Lustre Programming is Work**

As a very simple and classical example, the program shown below is a counter of “events”: It takes as inputs two Boolean flows “evt” (true whenever the counted “event” occurs), and “reset” (true whenever the counter should be reinitialized), and returns the number of occurrences of “events” which occurred since the last “reset”.

node Count(evt, reset: bool)returns(count: int);

let

count = if (true -> reset) then 0

else if evt then pre(count) + 1

else pre(count);

tel

Intuitively, “true -> reset” is a Boolean flow, which is true at the initial instant and whenever “reset” is true; when it is true, the value of “count” is 0; otherwise, when “event”

is true, “count” is incremented, otherwise it keeps its previous value.

Once declared, such a “node” can be used anywhere in a program, as a user-defined operator. For instance, our counter can be used to generate an event “minute” every 60 “second”, by counting “second” modulo 60:

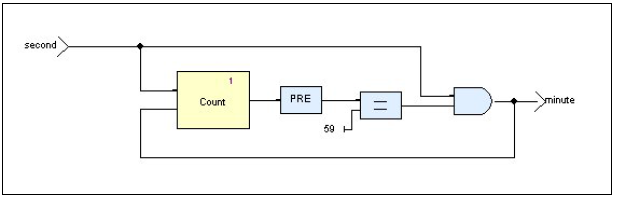


Figure 1: A graphical view in Scade

mod60 = Count(second, minute);

minute = second and pre(mod60)=59;

Here, “mod60” is the output of a “Count” node, counting “second”, and reset each “minute”, while “minute” is true whenever the “second” occurs when the previous value of

“mod60” is 59.

So, through the notion of node, Lustre naturally offers hierarchical description and component reuse. Data traveling along the “wires” of an operator network can be complex, structured information. From a temporal point of view, industrial applications show that several processing chains, evolving at different rates, can appear in a single system. LUSTRE offers a notion of boolean clock, allowing the activation of nodes at different rates.

Finally, one can express some knowledge about the input of a program using assertions.

These assertions are taken into account in verification (the desired property is only intended to hold when the inputs satisfy the assertion), for automatic testing (only input scenarios satisfying the assertion are generated), and sometimes for code optimization.

**5) Implementation**

Control synthesis

Clearly, boolean variables play an important role in LUSTRE: as clocks or conditions, they are often used to implement what is usually represented by control in imperative languages. Their computation must thus be carefully implemented. Let us illustrate how the rules of the dynamic semantics can be used to evaluate boolean expressions at compile time.

*Consider the following “program”, where b is an input variable:*

c = false -> b and not pref (c);

*Translation into basic syntax provides the program PO:*

c = false -> b and not pc:

pc = nil fby c:

Now, from the rules of dynamic semantics, we have

C:\Users\aiDaH\Desktop\2.PNG

where a(c) = false, I - nil and the program PI is as fol1ows:

c = b and not pc;

PC = false fby c;

Again from the rules of dynamic semantics, we get that l

* if the input b is false then

C:\Users\aiDaH\Desktop\3.PNG

where a(c) = false, o(pc) = false

* if the input b is true then

C:\Users\aiDaH\Desktop\4.PNG

where o(c) = true, afpc) = false and the program P2 is as follows:

c = b and not pc:

PC = true fby c;

Finally, we have that, whatever be the input b

,**C:\Users\aiDaH\Desktop\5.PNG**

**6) Conclusion**

As a conclusion the LUSTRE language, its main applications, and its associated tools have been presented. Asconcluding remarks, we will compare the LUSTRE approach with some alternative approaches, from both programming language and verification points of view.

**7) Reference**

1. Wikipedia, (2014). Lustre Programming Language. Obtained from <http://en.wikipedia.org/wiki/Lustre_%28programming_language%29>
2. LUSTRE/LESAR. Obtained from <http://www.cfdvs.iitb.ac.in/download/Docs/verification/tools/kronos/Verimag_Home_Page/SYNCHRONE/lustre-english.html>
3. Nicolas Halbwachs, Verimag, Grenoble (1993). The synchronous data-flow language Lustre.
4. Nicolas Halbwachs , Paul Caspi, Pascal Raymond and Daniel Pilaud (1991).The Synchronous Data Flow Programming Language LUSTRE.
5. F. Rocheteau and N. Halbwachs (1989). *Rex Workshop on Real-Time:*Implementing reactive programs on circuits, a hardware implementation of Lustre