The synchronous data-flow language Lustre

Nicolas Halbwachs
Verimag-CNRS
Grenoble – France

1 Introduction

The synchronous language Lustre was designed in the eighties, and resulted in the industrial software development tool Scade[1], which is now in use in many major companies developing embedded software (avionics, transportation, energy, ...). [Hal05] tells the story of Lustre and Scade.

Lustre is based on the synchronous paradigm [IEE91, Hal93, BCE+03]: the behaviour 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 *automaton*: the states are the valuations of the memory, and each reaction corresponds to a transition of the automaton. Such a transition may involve many computations, which, from the automaton point of view, are considered *atomic* (i.e., input changes are only taken into account between two reactions). This is the essence of the synchronous paradigm, where such a reaction is often said to *take no time*. An atomic reaction is called an *instant* (logical time), and all the events occurring during such a reaction are considered *simultaneous*. The way this logical time scale is defined, i.e., the way these reactions are triggered, is left to the environment: a program can be either event-triggered or time-triggered.

Synchronous languages aim at providing high level, modular, constructs, to make the design of such an automaton easier. The basic construct that all these languages provide, is a notion of synchronous concurrency, inspired by Milner's synchronous product [Mil81, Mil83]: in the sampling scheme, when automata are composed in parallel, a transition of the product is made of "*simultaneous*" transitions of all of them. When participating in such a compound transition, each automaton considers the outputs of others as being part of its own inputs. This "*instantaneous*" communication is called the *synchronous broadcast* [BCG88]. The important point is that, in contrast with the asynchronous concurrency considered in asynchronous languages like Ada, this synchronous product can preserve *determinism*, a highly desirable feature in reactive systems design.

Examples of other synchronous languages are Esterel [BG92, BS91], Signal [LGLL91, LTL03], ReactiveC [Bou91], SL [BdS96], Synchronous Lucid [Pou06], and ReactiveML [MP05].

[1] see www.esterel-technologies.com/products/scade-suite/

Beside 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 [Kah74, AW85].

2 An overview of Lustre

The initial descriptions of Lustre appeared in [CPHP87, HCRP91]. We briefly recall here the general principles of the language: A Lustre program operates on *flows* of values. Any variable (or expression) x represents a flow, i.e., an infinite sequence $(x_0, x_1, \ldots, x_n, \ldots)$ of values, x_n being the value of x at the nth reaction of the program. A program computes output flows from input flows. Output (and possibly local) flows are defined by means of equations (in the mathematical sense), an equation "x=e" meaning "$\forall n, x_n = e_n$". So, an equation can be understood as a temporal invariant. Lustre operators operate globally on flows: for instance, "x+y" is the flow $(x_0 + y_0, x_1 + y_1, \ldots, x_n + y_n, \ldots)$. In addition to usual arithmetic, Boolean, conditional operators — extended pointwise to flows as just shown — we will consider only two temporal operators:

- the operator "pre" ("*previous*") gives access to the previous value of its argument: "pre(x)" is the flow $(nil, x_0, \ldots, x_{n-1}, \ldots)$, where the very first value "*nil*" is an undefined ("non initialized") value.
- the operator "->" ("followed by") is used to define initial values: "x -> y" is the flow $(x_0, y_1, \ldots, y_n, \ldots)$, initially equal to x, and then equal to y forever.

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

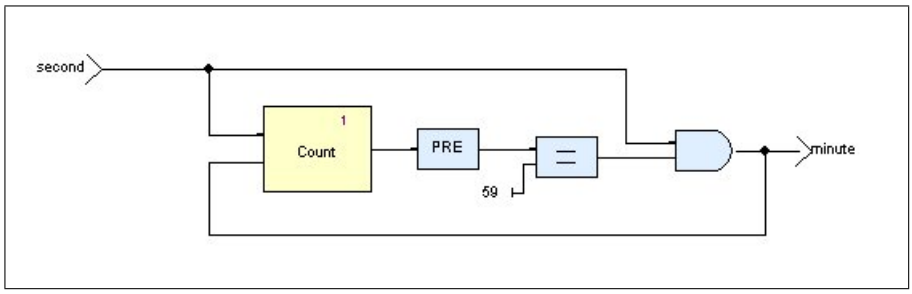


Figure 1: A graphical view in Scade

"second", by counting "second" modulo 60 :

```
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 informations.

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.

The graphical counterpart of Lustre textual syntax is obvious; for instance, Fig. 1 is a Scade view of the "minute detector" described before.

3 Available tools and papers

Some significant examples of Lustre programs have been published in [Hol94, CD96]. Apart from industrial tools provided by the Scade toolbox, academic tools (see `www-verimag.imag.fr/SYNCHRONE/`) consist of the compiler V4, the model-checker Lesar [HLR92], the automatic testing tool Lurette [RWNH98, JRB04], and translators from Simulink and Stateflow to Lustre [CCM+03]. Extensions of the language concern arrays [Mor02, MM04] and the combination with explicit automata [JLRM94, MR98]. Lucid-synchrone [CP95, CP96, Pou06] is a higher-order extension of Lustre. A lot of work has been devoted to the compilation to distributed or multi-thread code [GC92,

CMR01, CS00, SC04]. [BCDPV99] presents a methodology for proving Lustre programs with PVS [ORS92].

References

[AW85] E. A. Ashcroft and W. W. Wadge. LUCID, *the data-flow programming language*. Academic Press, 1985.

[BCDPV99] S. Bensalem, P. Caspi, C. Dumas, and C. Parent-Vigouroux. A methodology for proving control programs with Lustre and PVS. In *Proceedings of Dependable Computing for Critical Applications, DCCA-7, San Jose*. IEEE Computer Society, January 1999.

[BCE+03] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1), January 2003.

[BCG88] G. Berry, P. Couronné, and G. Gonthier. Synchronous programming of reactive systems, an introduction to ESTEREL. In K. Fuchi and M. Nivat, editors, *Programming of Future Generation Computers*. Elsevier Science Publisher B.V. (North Holland), 1988. INRIA Report 647.

[BdS96] F. Boussinot and R. de Simone. The SL synchronous language. *IEEE Transactions on Software Engineering*, 22(4):256–266, April 1996.

[BG92] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[Bou91] F. Boussinot. Reactive C: An extension of C to program reactive systems. *Software Practice and Experience*, 21(4):401–428, 1991.

[BS91] F. Boussinot and R. de Simone. The ESTEREL language. *Proceedings of the IEEE*, 79(9):1293–1304, September 1991.

[CCM+03] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert. From Simulink to Scade/Lustre to TTA: A layered approach for distributed embedded applications. In *LCTES 2003*, San Diego, CA, June 2003.

[CD96] T. Cattel and G. Duval. The steam boiler problem in Lustre. Available by FTP at `ltidec1.epfl.ch:/pub/lustre/steamboiler_lustre.ps.gz`, 1996.

[CMR01] P. Caspi, C. Mazuet, and N. Reynaud Paligot. About the design of distributed control systems, the quasi-synchronous approach. In *SAFECOMP'01*. LNCS 2187, 2001.

[CP95] P. Caspi and M. Pouzet. A functional extension to LUSTRE. In *Eighth International Symp. on Languages for Intensional Programming, ISLIP'95*, Sidney, May 1995.

[CP96] P. Caspi and M. Pouzet. Synchronous Kahn networks. In *Int. Conf. on Functional Programming, Philadelphia*. ACM SIGPLAN, May 1996.

[CPHP87] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. LUSTRE: a declarative language for programming synchronous systems. In *14th ACM Symposium on Principles of Programming Languages, POPL'87*, Munchen, January 1987.

[CS00] P. Caspi and R. Salem. Threshold and bounded-delay voting in critical control systems. In *FTRTFT'2000*, Pune, India, September 2000. LNCS 1926.

[GC92] A. Girault and P. Caspi. An algorithm for distributing a finite transition system on a shared/distributed memory system. In *PARLE'92, Paris*, July 1992.

[Hal93] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Pub., 1993.

[Hal05] N. Halbwachs. A synchronous language at work: the story of Lustre. In *Third ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE'2005*, Verona, Italy, July 2005.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow programming language LUSTRE. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*, pages 785–793, September 1992.

[Hol94] L. Holenderski. Production cell in LUSTRE. In C. Lewerentz and Th. Lindner, editors, *Case Study "Production Cell": a Comparative Study in Formal Software Development*, Forschungszentrum Informatik, Karlsruhe, 1994. FZI-Publikation 940001, ISSN 0944-3037.

[IEE91] Another look at real-time programming. *Special Section of the Proceedings of the IEEE*, 79(9), September 1991.

[JLRM94] M. Jourdan, F. Lagnier, P. Raymond, and F. Maraninchi. A multiparadigm language for reactive systems. In *5th IEEE International Conference on Computer Languages*, Toulouse, May 1994. IEEE Computer Society Press.

[JRB04] E. Jahier, P. Raymond, and P. Baufreton. Case studies with Lurette V2. In *First International Symposium on Leveraging Applications of Formal Method, ISoLa 2004*, Paphos, Cyprus, October 2004.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP 74*. North Holland, 1974.

[LGLL91] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, September 1991.

[LTL03] P. Le Guernic, J.-P. Talpin, and J.-C. Le Lann. Polychrony for system design. *Journal for Circuits, Systems and Computers, Special Issue on Application Specific Hardware Design*, April 2003.

[Mil81] R. Milner. On relating synchrony and asynchrony. Technical Report CSR-75-80, Computer Science Dept., Edimburgh Univ., 1981.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *TCS*, 25(3), July 1983.

[MM04] F. Maraninchi and L. Morel. Arrays and contracts for the specification and analysis of regular systems. In *Fourth International Conference on Application of Concurrency to System Design (ACSD)*, Hamilton, Ontario, Canada, June 2004.

[Mor02] Lionel Morel. Efficient compilation of array iterators for Lustre. In Florence Maraninchi, Alain Girault, and Éric Rutten, editors, *Electronic Notes in Theoretical Computer Science*, volume 65. Elsevier, 2002.

[MP05] L. Mandel and M. Pouzet. ReactiveML, a reactive extension to ML. In *ACM International Conference on Principles and Practice of Declarative Programming (PPDP)*, Lisboa, July 2005.

[MR98] F. Maraninchi and Y. Rémond. Mode-automata: About modes and states for reactive systems. In *European Symposium On Programming*, Lisbon (Portugal), March 1998. Springer Verlag.

[ORS92] S. Owre, J. Rushby, and N. Shankar. PVS: a prototype verification system. In *11th Conf. on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer Verlag, 1992.

[Pou06] Marc Pouzet. *Lucid Synchrone, version 3. Tutorial and reference manual*. Université Paris-Sud, LRI, April 2006. Distribution available at: `www.lri.fr/~pouzet/lucid-synchrone`.

[RWNH98] P. Raymond, D. Weber, X. Nicollin, and N. Halbwachs. Automatic testing of reactive systems. In *19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[SC04] N. Scaife and P. Caspi. Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In *Euromicro conference on Real-Time Systems (ECRTS'04)*, Catania, Italy, June 2004.