6

A High-Level Language for Programming Complex Temporal Behaviors and Its Translation into Synchronous Circuits

Ching-Tsun Chou[1], *Jiun-Lang Huang*[2], *and Masahiro Fujita*[1]
[1] *Fujitsu Laboratories of America*
3350 Scott Blvd., Bldg. 34, Santa Clara, CA 95054, U.S.A.
E-mail: {ctchou,fujita}@fla.fujitsu.com
[2] *Dept. of Electrical and Computer Engineering*
Univ. of California Santa Barbara, Santa Barbara, CA 93106, U.S.A.
E-mail: lang@redwood.ece.ucsb.edu

Abstract

We present YASL, a synchronous programming language that supports high-level programming of finite-state machines by providing a rich set of control constructs for specifying complex temporal behaviors and an automatic procedure for translating such high-level temporal specifications into finite-state machines in the form of synchronous circuits. In addition to operators for expressing watchdog, conditional, sequencing, and concurrency, YASL offers a powerful *temporal projection* operator that supports hierarchical description of temporal behaviors at different granularities of time.

This is a summary of a longer paper, in which we describe the syntax and semantics of YASL in detail, explain how to translate it into synchronous circuits without schizophrenia, and compare it with related languages Tempura, Esterel, and Handel. To obtain the full version, please write to the first author.

Keywords

Complex temporal behaviors, synchronous languages, circuit translation.

1 SUMMARY

The complex temporal behaviors of hardware are implemented by finite-state machines (FSMs). As is well-known, unstructured FSMs are difficult to understand and construct, since even minor changes in the state-transition structure of a FSM can lead to profound and often unexpected changes in its behavior. While an unstructured FSM with 10 states is probably within the grasp of the human mind, an unstructured FSM with 100 states is almost certainly beyond it. In order to make the programming of FSMs easier, it is desirable to have a high-level programming language for FSMs with the following characteristics:

- It allows programs to be combined in arbitrary ways using such natural operations as concurrency, sequencing, iteration, conditional, preemption, and suspension.
- It supports *temporal abstraction*—the hierarchical description of temporal behaviors at different granularities of time [3].
- It has a simple and precise semantics.
- Its programs are directly synthesizable into hardware.

In this paper we present such a language, which we call YASL (Yet Another Synchronous Language) because its semantics is based on the same *synchrony hypothesis* as adopted by other synchronous languages [2], which assumes that every program is so fast in response to stimuli from its environment (i.e., other programs) that such response can be considered instantaneous. The reason for our adopting the synchrony hypothesis is simple: most hardware systems are clocked and, in a clocked system, one does not care what happens during a clock cycle other than the end results (i.e., the values that are latched into registers at the end of the clock cycle).

The control constructs of YASL are inspired by Tempura [4], an executable subset of interval temporal logic. They allow programs to be combined in arbitrary ways using watchdog (**await** e), conditional (**if** e **then** p **else** q), sequencing ($p\,;q$), concurrency ($p \parallel q$), and temporal projection ($p \lhd q$). The *temporal projection* operator $\lhd$ is the most distinctive feature of YASL. The program $p \lhd q$ behaves like p at the beginning. If p terminates then, $p \lhd q$ also terminates without executing q; otherwise q is executed and p is suspended until q terminates, whereupon p is resumed. If p terminates then, $p \lhd q$ terminates; otherwise q is re-executed, p is suspended until q terminates, and the same process is repeated again. This is illustrated in Figure 1, where the dashed arrows depict flow of control, the bullets represent execution steps of p and q, and two adjacent bullets indicate that the two steps of q (and the intervening step of p) should be executed in the same clock cycle.

There are two ways of looking at a temporal projection $p \lhd q$. First, it can be viewed as a generalized iteration where p is used to control the iteration

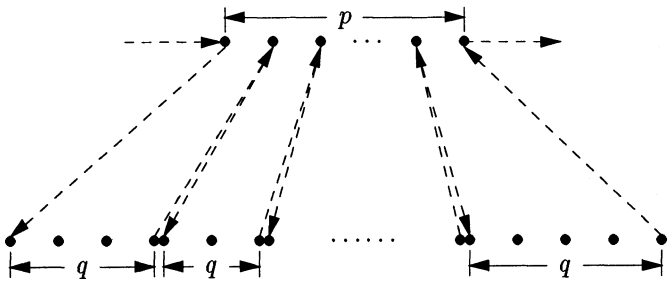

Figure 1 How to execute $p \lhd q$

of q. Indeed, the usual looping constructs can be easily derived from $\lhd$:

$$\textbf{while}\ e\ \textbf{do}\ q \triangleq \textbf{await}(\neg e) \lhd q$$

$$\textbf{repeat}\ q\ \textbf{until}\ e \triangleq (\textbf{pause}; \textbf{await}(e)) \lhd q$$

where **await**(e) terminates once e is true and **pause** pauses for one step and then terminates. Second, it can be viewed as a form of *temporal abstraction* [3] where p is executed using the slower clock defined by the endpoints of q's executions. For example, the suspension operator of Esterel [1] can be easily defined using $\lhd$:

$$\textbf{suspend}\ p\ \textbf{when}\ e \triangleq p \lhd (\textbf{pause}; \textbf{await}(\neg e))$$

where p in effect uses the negation of e as its clock. Furthermore, it can be shown that $\lhd$ is associative, so the temporal projections can be stacked one on top of another to form a hierarchical description of temporal behaviors at different granularities of time. For example, in many bus protocols, such as the Pentium Pro processor bus protocol [5], a memory or I/O *operation* may consist of multiple *transactions*, each of which consists of multiple *phases*, each of which takes multiple clock cycles. A controller for a bus agent might use the following code fragment:

```
OPERATION ◁ TRANSACTION ◁ PHASE
```

We developed the precursor of YASL and its circuit translation without knowledge of the body of works on synchronous languages [2]. Since we became aware of it, we have been influenced by the synchronous language Esterel [1]. In particular, we have borrowed the notion of *signals* and the style of defining formal semantics from Esterel. Interestingly, though they were developed independently, the control constructs of YASL and Esterel are equally expressive in the sense that they can easily simulate each other.

REFERENCES

[1] G. Berry, *The Constructive Semantics of Pure Esterel (draft version 2.0)*, 22 May 1996. (Available from `http://cma.cma.fr/Esterel/`)

[2] N. Halbwachs, *Synchronous Programming of Reactive Systems*, Kluwer Academic Publishers, 1993.

[3] T. F. Melham, "Abstraction Mechanisms for Hardware Verification", pp. 267–291 of G. Birtwistle and P.A. Subrahmanyam (ed.), *Current Trends in Hardware Verification and Automated Theorem Proving*, Springer-Verlag, 1989.

[4] B. Moszkowski, *Executing Temporal Logic Programs*, Cambridge University Press, 1986.

[5] *Pentium Pro Family Developer's Manual, Vol. 1: Specifications*, Intel Corporation, 1996.