TEMPORAL LOGIC - A TOOL FOR SYSTEM MODELING

UDK 519.711.3

Bogdan Dugonik
Technical Faculty Maribor

Abstract

This paper presents a model for processes, that communicate by message-passing, and the use of temporal logic for their description. The specification of process determines behaviour on its external ports, whence its internal behaviour is not significant. Specifications based on the model are compositional, that is, we get external behaviour of a system from specifications of its components. In the introduction a process recorded with traces is presented. A model is shown that represents a process specification by use of temporal logic. Temporal logic operators, examples of temporal assertions and some specifications for primitive and composed processes are given.

Povzetek

V članku je prikazan model za procese in mreže procesov, ki komunicirajo izključno s prehajanjem sporočil, in uporaba časovne logike za njihov opis. Specifikacija procesa določa obnašanje na njegovih zunanjih priključkih, dogajanje v njegovi notranjosti pa nas ne zanima. Opisi procesov so komponibilni, kar pomeni, da zunanje obnašanje mreže dobimo iz specifikacij njenih komponent. Pri tem prikrijemo njeno notranjo strukturo. V uvodu predstavimo zapis procesa s sledmi. Prikažemo tudi specifikacijo procesa in mreže procesov z uporabo časovne logike. Opisani so operatorji časovne logike, podanih je nekaj primerov časovnih izjav ter specifikacij enostavnih in sestavljenih procesov.

Introduction

This paper treats processes that communicate exclusively through message passing. Messages are transmitted over input and output ports. Let us observe a process on its external ports for a time, up to some moment. We get a sequence of events. The sequence gives us a trace. The trace is an information about behaviour of the process up to that moment. Traces are appropriate to describe only terminating processes. The model of a system is modular, it is composed of some primitive processes. In a system output ports of processes are linked to inputs of another processes. Some input and output ports are not linked. Nonlinked ports are external ports and linked ports are internal ports.

A process is determined by a set of possible input-output behaviours . The behaviour on a set of input and output ports is an infinite sequence of observations :

$$\sigma = (t_0, In_0, Out_0, Rd_0), (t_1, In_1, Out_1, Rd_1), \ldots$$

t is a trace of events on input ports (input events) and events on output ports (output events).
Events represent communication between processes. Communication can be synchronous or asynchronous. If communication is synchronous, a process cannot send a message until the receiving process is ready to read the message on his input ports. If communication is asynchronous, a process can send message on his output ports as soon as it is ready without having to wait for the receiving process. In case of synchronous communication, input events present the data read by a process, and in case of asynchronous communication, they present the data that have appeared on input ports. Output events in both ways of communication present messages sent by a process at its output ports. An event is recorded in the trace as a pair (x,k), where x is a datum and k is the port name where the event has appeared. In and Out are boolean functions. If they are true for input - output ports, then we say the process is ready to

communicate. If logical values of some input port functions are true, then we say the process is ready to read on those ports. If logical values of some output port functions are true, then we say the process is ready to send on those output ports . Rd gives us the number of messages read up to the observed moment for each input port . It is clear that this number cannot be greater than the number of messages which have appeared on an input port. We could describe the process by writing down all possible sequences σ, which is vague and for more complicated cases practically impossible. Because of that , we apply a language which enables us to express all possible behaviours of a process.

Temporal logic:

Natural languages are very expressive, but not precise, whereas formal languages are very precise but not so expressive. Formal languages have strictly defined semantics and syntax. It is impossible to say everything in them. But what you can say is unambiguous.

Temporal logic is a formal language. It allows us to make a temporal description of a process and also formal dealing of it. We present a model of temporal logic with infinite sequence of states

$$\sigma = s_0, s_1, s_2, \ldots$$

Assertions in temporal logic have a temporal meaning because they include some temporal operators. Language employs a set of basic symbols: individual variables and constants, function and predicate symbols. These are divided into two subsets: a set of local symbols and a set of global symbols.
Global symbols keep their values unchanged from state to state. Local symbols can vary their values from one state to another. We use boolean operators, classical operators and temporal operators. Existence quantification and universal quantification are also included in the system.

The operators of temporal logic:

By w we denote a term or a formula that may contain some temporal operators.

- temporal operator "**always**":

  $\Box$ w means: w is true in this moment and will be true forever.

- temporal operator "**eventually**":

  $\Diamond$ w means: w will be true at least in one of following moments.

- temporal operator "**until**":

  $w_1 \mathcal{U} w_2$ means : w_1 is continually true until w_2 becomes true, and w_2 does indeed become true.

- temporal operator "**unless**":

  $w_1 \mathcal{W} w_2$ means: w_1 is continually true or there is a moment, when w_2 is true and up to the moment w_1 is continually true.

- temporal operator "**next**":

  $\circ$ means: w will be true in next moment.

Temporal assertions:

Formulas given below are examples of some temporal assertions. They are compounded by using temporal operators mentioned above. We will write $\wedge$ for conjunction, $\neg$ for negation, and $\Rightarrow$ for implication.

$\Box(u \Rightarrow \Diamond v)$	- whenever u is true, it will eventually be followed by v.
$\Diamond \Box w$	- eventually w will become true and it will remain thrue forever.
$\Box \Diamond w$	- every instant is followed by some later instant, when w is true, so w is true indefinitely often.
$\Diamond v \Rightarrow ((\neg v) \mathcal{U} u)$	- if v ever happens, its first occurrence is preceded by, or coincides with u.

We choose some fixed reference moment named reference moment when interpreting a temporal assertion. It tells us, what holds at the reference moment, i.e. now, and what will happen in the future of it. It says nothing about what happened before. Instead of $\sigma = s_0, s_1, s_2, \ldots$ let us write $\sigma = \sigma_0, \sigma_1, \sigma_2, \ldots$ where σ_i is an observation in behaviour σ . An observation can be treated as a state. Usually, a state of a process reflects current values of variables of the process. Hence, an observation σ_i is a state, which involves the history of process up to a current moment in its trace t_i. Specification of a process signifies how this process behaves. For this model we write process specifications in the form of temporal assertions.
The specification of the system P has the form <P> R , where R is an assertion, written in temporal logic. Specification <P> R is read: each behaviour of system P satisfies assertion R. <P> R is external specification, if R specifies behaviour on P's external ports. System specification is a conjunction of component assertions. The external specification is obtained by using a proof system.
The proof system for specifications includes rules for temporal logic, rules regarding the description of the domain of values, rules for designing a system, a set of system components with their precise specifications, and axioms which define the behaviour of the system. Specification defines two kinds of process properties: safety and liveness. Informally, safety specification asserts, what the process may do, and liveness specification asserts, what it must do.We will give some examples of specifications.

First, we show how certain ports of a process can be disabled, discarded from communication by using negation operator $\neg$ with In or Out function.
In the specification below input port a_1 and output port b_1 are disabled. Input a1 can not read the data, and b1 can not send the data.

$$\langle C \rangle \quad \Box \neg \mathrm{In}(a_1) \;\wedge\; \Box \neg \mathrm{Out}(b_1)$$

Now, let us assume that communication is asynchronous. We will write external specifications of processes. That is, we will specify a process only by means of lengths of ports' traces, without using In, Out and Rd. By $|b|$ we denote the length of trace on port b. $b \sqsubseteq c$ means, that sequence b is a prefix of sequence c. First example below (fig.1.) gives a process with input port k and output port l. The process reads six values on input k and writes them on output l leaving out the first two values.

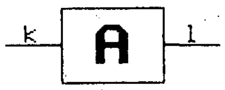

fig.1. Process with input port k and output port l.

$$\langle A \rangle \ \Box\, l \sqsubseteq [\ k(2),k(3),k(4),k(5)\ ]$$

$$\wedge\ (\ |l| = 0 \ \mathcal{U}\ |k| \geqq 3)$$

$$\wedge\ (\Diamond\ |k| = 3 \ \Rightarrow\ \Diamond\ |l| = 1)$$

$$\wedge\ (\Diamond\ |k| = 4 \ \Rightarrow\ \Diamond\ |l| = 2)$$

$$\wedge\ (\Diamond\ |k| = 5 \ \Rightarrow\ \Diamond\ |l| = 3)$$

$$\wedge\ (\Diamond\ |k| \geqq 6 \ \Rightarrow\ \Diamond\ |l| = 4)$$

The first line is safety specification. It indicates that at most four values may occur on the output port, which will be read on input port k. The second conjunct indicates that the length of the output trace is zero, unless the length of input trace is three. The third conjunct and those following require the length of the output trace to be increased by one as soon the length of the input trace is increased.

Second example (fig. 2) is a process with two inputs and one output. It reads one value on j and one value on k. Then it writes first the value on the output l from input k, and later the value from input j.

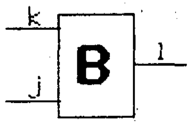

Fig.2. A process named B with two inputs and one output

$$\langle B \rangle \ \Box\, l \sqsubseteq [\ k(0),j(0)\ ]$$

$$\wedge\ (\ \Diamond\ |k| = 1 \ \Rightarrow\ \Diamond\ |l| = 1\ )$$

$$\wedge\ (\ \Diamond(\ |j| = 1 \wedge |k| \geqq 1) \ \Rightarrow\ \Box\ |l| = 2)$$

Finally, we present an asynchronous network P of component processes A,B and C. Components A and B have one input and one output, component C two inputs and one output. Components are linked as shown in figure 3. Process A reads one value on input a and writes it twice on output c. Process B reads one value on input b and writes it once on output d. Process C reads values on its inputs, then writes them on output e as follows: first a value from input d, then both values from input c.

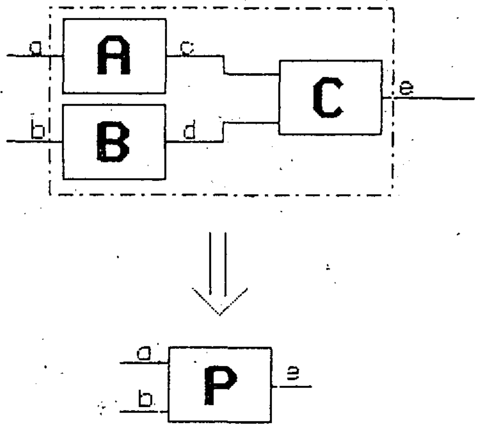

Fig.3. Process P with visible and hidden internal structure.

First, specifications for each components are given:

$$\langle A \rangle \quad \Box\, c \sqsubseteq [\ a(0),a(0)\ ]$$

$$\wedge\ (\ \Diamond\ |a| \geqq 1 \Rightarrow \Diamond\Box\ |c| = 2)$$

$$\langle B \rangle \quad \Box\, d \sqsubseteq [\ b(0)\ ]$$

$$\wedge\ (\ \Diamond\ |b| = 1 \Rightarrow \Diamond\Box\ |d| = 1)$$

$$\langle C \rangle \quad \Box\, e \sqsubseteq [\ d(0),c(0),c(1)\ ]$$

$$\wedge\ (\ \Diamond\ |d| \geqq 1 \Rightarrow \Diamond\ |e| = 1)$$

$$\wedge\ (\ \Diamond\ (\ |c| = 1 \wedge |d| \geqq 1) \Rightarrow (\ \Diamond\ |e| = 2)$$

$$\wedge\ (\ \Diamond\ (\ |c| = 2 \wedge |d| \geqq 1) \Rightarrow (\ \Diamond\ |e| = 3)$$

External specification of P is obtained from conjunction of component specifications by using proof rules.

$$\langle P \rangle \quad \Box\, e \sqsubseteq [\ b(0),\ a(0),\ a(0)\ ]$$

$$\wedge\ (\ \Diamond\ |b| \geqq 1 \Rightarrow \Diamond\ |e| = 1\ )$$

$$\wedge\ (\ \Diamond\ (\ |a| \geqq 1 \wedge |b| \geqq 1) \Rightarrow \Diamond\Box\ |e| = 3\ )$$

It tells us how the system behaves on its external ports.

Conclusion

We presented a model for processes which uses temporal logic as one of existing formal languages for specifying them. The benefit of temporal logic is its ability of describing temporal behaviour of processes. Specifications can be short and so expressive, that they need no extra comments. Troubles arise, if we want to set the exact time, because in the model we can express ourselves only qualitatively about the time of events.

References:

[1] Dugonik B., Modeliranje procesov, diplomsko delo, Tehniška fakulteta v Mariboru, 1987

[2] Hoare C.A.R., Communicating Sequential Processes , Prentice - Hall International, Ltd., London 1985.

[3] Kapus T., časovna logika in mreže procesov, diplomsko delo, Tehniška fakulteta v Mariboru, 1987

[4] Lamport L., What Good is Temporal Logic ?, Information Processing (1983),pp. 657-668

[5] Manna Z.,Pnueli A., Verification of Concurrent Programs, Part I: The temporal Framework; Tehnical Report STAN-SC-81-836, Standford University, June 1981.

[6] Nguyen V., Demers A., Gries D., Owicki S., A model and temporal proof system for networks of processes, Distributed Computing (1986), 1, pp. 7-25.