1997-12-01

18th IEEE Real-Time Systems Symposium: Work in Progress Sessions

Compositional Reasoning about Real-Time Asynchronous Communication with Time-Outs*

D. Peticolas

Department of Computer Science
University of California, Davis
Davis, CA 95616

F. A. Stomp

Department of Computer Science
University of California, Davis
Davis, CA 95616

Abstract

This paper describes ongoing work in developing a compositional trace-based semantics and proof system for a real-time language. The semantics models distributed processes communicating over asynchronous FIFO communication channels. Sending processes can specify time-out *periods for individual messages. Messages not received within their time-out period are 'lost'. Program behavior is modeled as traces of events, including events (such as asynchronous messages) which occur after termination. The proof system uses specification triples with explicit variables for time and program traces.*

1 Introduction

This paper describes ongoing work in developing a compositional trace-based semantics and proof system for a real-time language. The semantics models distributed processes communicating over asynchronous FIFO communication channels. Sending processes can specify *time-out* periods for individual messages. Messages not received within their time-out period are 'lost'.

Additional work in real-time asynchronous message-passing includes [4, 2]. These works model asynchronous message passing using explicit data structures such as FIFO queues. In contrast, the approach presented here uses a different method – associating timestamps with individual messages [7] and modeling messages as events which occur in the 'future', i.e., *after* a send command has completed execution.

Using a trace semantics with future events eliminates the need for additional data structures to model communications channels. Furthermore, the semantics of message time-out can also be defined in a straightforward, compositional way.

Our method is used to define a denotational semantics for a CSP-like language in the style of the synchronous real-time semantics of [6, 8]. Due to space requirements, the presentation will be at an informal level, with formal definitions omitted. A more formal presentation will be found in a forthcoming paper [12].

*This work was supported by DARPA under contract N00014-93-1-1322 with the Office of Naval Research.

2 Language

The programming language includes a guarded command (with an optional time-out arm), a delay statement, iteration, and parallel composition. Variables are not shared between processes. Processes communicate along unidirectional asynchronous channels which link two distinct processes and messages are received in the order sent. However, messages have a sender-specified time-out period and messages that are not received within their time-out are removed from their channel and are lost. We will use the term FIFO to denote traditional first-in first-out ordering modulo any lost messages.

The syntax of the programming language is given below where c, c_i are channels, x, x_i are variables, n is a natural number, and b_i and e are boolean expressions and numerical expressions, respectively.

$$P ::= \textbf{skip} \mid \textbf{delay}\ e \mid x := e \mid c!\langle e_1, e_2\rangle \mid c?x \mid P_1; P_2 \mid G \mid *G \mid P_1 \| P_2$$

$$G ::= [\ []_{i=1}^{n}\ b_i \to S_i\ ] \mid [\ []_{i=1}^{n}\ b_i; c_i?x_i \to S_i\ []\ b_{n+1}; \textbf{delay}\ e \to S_{n+1}\ ]$$

Since one of the contributions of this work involves describing the semantics of sending messages with time-outs and composition, we describe the send command first, and then briefly describe the receive command and parallel composition. In the discussion below, several constants of the form C_{sub} are used to denote strictly positive time durations.

Koymans *et al* [9] divide an asynchronous message transmission into five stages. Only the first stage involves the actual send command the rest are waiting for transmission, transmission, arrival, and receipt. Later work by other authors [4, 16] develops a real-time compositional trace semantics and proof system for asynchronously communicating processes based on Hooman's

work for synchronous real-time processes [6]. However, these semantics assume zero transmission time, i.e., a message can be received immediately upon termination of the send command.

We generalize asynchronous sending to the more realistic case of a non-zero transmission time. Furthermore, we allow senders to specify a time-out period for individual messages. The command $c!\langle e_1, e_2\rangle$ (asynchronously) sends the message e_1 on channel c with a time-out of e_2 time units. All send commands execute for $\mathcal{C}_{send}$ time units.

A message becomes *visible* (and therefore receivable) to the receiving process $\mathcal{C}_{xmit}$ time units after the send command has begun execution. Thus, $\mathcal{C}_{xmit}$ includes both the send command execution and the transmission time. Once a message becomes visible to a receiver, it remains visible until one of two events occur:

- The message is received, or
- The time-out period elapses and the message is lost.

The informal meaning of some other commands is given below:

- $c?x$ blocks (possibly forever) until a message is visible on channel c and then transfers the message into x. Transferring the message requires $\mathcal{C}_{recv}$ time units. If more than one message is visible, the message which has been visible for the longest time is received. In other words, messages are received in FIFO order.
- $P_1 \| P_2$ is the parallel execution of P_1 and P_2 in which each process executes on a separate processor. This is the *maximal parallelism* assumption of [13] used in [8, 6, 16].

3 Semantics

The semantics employ the concepts of *states*, *events*, *traces*, and *models*. A state is a mapping from program variables to values. The value of an expression e in state σ is denoted by $\sigma(e)$.

An event is an observation of program activity at some point in time. The events used to model communication activity are listed below.

- $c??$ indicates that a process is waiting for a message on channel c.
- $c!!\langle v, \tau\rangle$ indicates the presence of a visible message on channel c with value v and which became visible at time τ. The time τ is used to specify that messages are received in the order they appear, as well as to distinguish between different occurrences of the same message [7].
- $\langle c, v, \tau\rangle$ indicates the receipt of v on channel c and v was first visible at time τ.
- $c \star \langle v, \tau\rangle$ indicates the time-out of v on channel c and v was first visible at time τ.

A trace is a pair $\langle t, \tau\rangle$ consisting of a (possibly infinite) time τ and a mapping t from continuous time values to sets of events. Function t represents the events occurring over time and τ represents the time at which program execution stops. Certain events may be observed in the future, after a process has finished execution. Specifically, the effect of a send command (a visible message and eventual communication or time-out) is represented by future events.

A model is a pair $\langle \sigma, t\rangle$ where t is a trace and σ is a state. It represents a computation of a program where σ determines the value of all variables when the program terminates. If t has infinite length, then σ is irrelevant. The meaning of programs is defined as a mapping from an 'initial' model to sets of 'final' models. The set of final models represents the possible computations of the program when executed following the initial model. We give definitions below for the send command and for parallel composition.

The meaning of a send command is defined as the union of two possibilities:

- The message is received before e_2 time units have elapsed, or
- The message is visible for e_2 time units and is then removed from the communications medium, i.e., the message is lost.

The meaning of a send command contains a trace for every possible time of communication for the first possibility and one trace for the second.

The parallel composition of two programs is essentially the parallel combination of each program's respective traces. However, we must exclude parallel combinations which cannot occur using three conditions: First, a process executing a receive statement never blocks when there are visible messages (maximal parallelism). Second, when two programs are executed in parallel, their traces must be consistent with respect to communications along their shared channels. Finally, a process always receives messages in FIFO order.

The semantics of parallel composition is then defined as every possible parallel execution of traces subject to the three requirements above.

4 Proof System

The proof system uses specification triples $\{p\}\, S\, \{q\}$ to express properties of programs. Postcondition q is interpreted over all models, including non-terminating computations.

As in [6], the assertion language uses a special variable, here denoted $\mathcal{T}$, to represent time. In addition, there is a special trace variable Π which denotes the continuous trace of events in a program. Thus, the assertion $c?? \in \Pi(5)$ is interpreted as "there is a process waiting on channel c at time 5."

The use of Π is in contrast to [6] in which the trace is implicit in the assertion language. Using an explicit trace variable permits backwards proof rules using syntactic substitution of trace expressions.

Below is the proof rule for the send command. The operator $\ominus$ is the pointwise union of trace expressions. The trace expression $send(c, e_1, e_2, \tau)$ is interpreted as the continuous sequence of events that would be observed in which a message was sent at time 0 and received (or lost) at time τ.

$$\frac{p \rightarrow \forall \tau \in [\mathcal{T} + \mathcal{C}_{xmit}, \mathcal{T} + \mathcal{C}_{xmit} + e_2] \; : \; q[(\mathcal{T} + \mathcal{C}_{send})/\mathcal{T}, \; (\Pi \oplus send(c, e_1, e_2, \tau))/\Pi]}{\{p\}\, c!\langle e_1, e_2\rangle \,\{q\}}$$

The proof rule for the receive command is very similar to that for send.

$$\frac{p \rightarrow \forall v, \tau_0, \tau \in [\mathcal{T}, \infty] : \; q[(\mathcal{T} + \mathcal{C}_{recv})/\mathcal{T}, \; (\Pi \oplus recv(c, v, \tau, \tau_0))/\Pi, v/x]}{\{p\}\, c?x \,\{q\}}$$

The proof rule for parallel composition is given below. As in [17], the rule is only valid if all references to Π in the assertions are in a restricted form Π_{cset} where $cset$ is a collection of channel names. In essence, for the rule to be valid, the restricted channels of q_i must be a subset of those in S_i to prevent interference between assertions. There is a similar restriction on the variables of q_i.

$$\frac{\{p_1\}\, S_1 \,\{q_1\}, \; \{p_2\}\, S_2 \,\{q_2\}}{\{p_1 \wedge p_2\}\, S_1 \parallel S_2 \,\{q\}},$$

for $q \equiv \exists t_1, t_2 . q_1[t_1/\mathcal{T}] \wedge q_2[t_2/\mathcal{T}] \wedge \mathcal{T} = max(t_1, t_2)$.

There is also an axiom for parallel composition which introduces the three conditions for parallel composition explained in Section 3. As in [6], these conditions are known as *well-formedness* conditions.

$$\{true\}\, S_1 \parallel S_2 \,\{WF_{cset}\},$$

where $WF_{cset} \equiv NoWait_{cset} \wedge Fifo_{cset} \wedge Agree_{cset}$ and $cset \subseteq chan(S_1 \parallel S_2)$.

The $NoWait_{cset}$ assertion specifies that a process never waits to receive on a channel $c \in cset$ while a message is visible on c. $Fifo_{cset}$ asserts that messages are received in the order sent and $Agree_{cset}$ asserts that S_1 and S_2 'match' on communications along channels in $cset$. As an example, $NoWait_{cset}$ is defined as:

$$\forall c \in cset, v, \tau, \tau_0 . \; c!!\langle v, \tau\rangle \in \Pi(\tau_0) \Rightarrow c?? \notin \Pi(\tau_0)$$

To combine the proof rule and axiom for parallel composition, it is useful to have proof rules for conjunction and consequence.

$$\frac{\{p_1\}\, S \,\{q_1\}, \{p_2\}\, S \,\{q_2\}}{\{p_1 \wedge p_2\}\, S \,\{q_1 \wedge q_2\}}$$

$$\frac{p \rightarrow p', \; \{p'\}\, S \,\{q'\}, \; q' \rightarrow q}{\{p\}\, S \,\{q\}}$$

Finally, we present a few of the more basic rules and axioms. Below are the axioms for **skip**, **delay**, and assignment.

$$\{p\}\, \mathbf{skip} \,\{p\}$$

$$\{p[(\mathcal{T} + e)/\mathcal{T}] \wedge \mathcal{T} < \infty\}\, \mathbf{delay}\ e \,\{p\}$$

$$\{p[(\mathcal{T} + \mathcal{C}_{assign})/\mathcal{T}, e/x] \wedge \mathcal{T} < \infty\}\, x := e \,\{p\}$$

The rule for sequential composition is straightforward:

$$\frac{\{p\}\, S_1 \,\{r\}, \{r\}\, S_2 \,\{q\}}{\{p\}\, S_1; S_2 \,\{(r \wedge time = \infty) \vee q\}}$$

Due to space requirements, we have omitted some of the rules, such as the rules for guarded commands.

5 Conclusion and Future Work

This paper has presented ongoing work in developing a compositional semantics and proof system for asynchronous message passing with time-outs. The work models program behavior as traces of events, including events that occur after program termination (such as asynchronous messages). The proof system uses specification triples with explicit variables for time and program traces.

Other work in real-time formal methods research includes predicate transfer with time-outs on sending and receiving [3], and proof outlines in real-time shared variable settings [15, 14]. A process algebraic approach including resources and priorities is used in [5], while state machines are used in [10, 1].

Future work will involve finishing the soundness proof and developing a completeness proof for the system. We are also mechanizing the system in the PVS theorem prover [11].

References

[1] R. Alur and D. Dill. The theory of timed automata. In *Real-Time Theory in Practice*, LNCS 600, pages 45–73. Springer-Verlag, 1992.

[2] J.C.M. Baeten and J.A. Bergstra. Real space process algebra. *Formal Aspects of Computing*, 5:481 529, 1993.

[3] A.J. Bernstein. Predicate transfer and timeout in message passing. *Information Processing Letters*, 24:43 52, 1987.

[4] F. de Boer and J. Hooman. The real-time behaviour of asynchronously communicating processes. In *Proc. 2nd Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 571, pages 451–472. Springer-Verlag, 1991.

[5] R. Gerber and I. Lee. Specification and analysis of resource-bound real-time systems. In *Real-Time Theory in Practice*, LNCS 600, pages 371–396. Springer-Verlag, 1992.

[6] J. Hooman. *Specification and Compositional Verification of Real-Time Systems.* LNCS 558. Springer-Verlag, 1991.

[7] R. Koymans. Specifying message passing systems requires extending temporal logic. In *Temporal Logic in Specification*, LNCS 398, pages 213–223. Springer-Verlag, 1989.

[8] R. Koymans, R.K. Shyamsundar, W.P. de Roever, R. Gerth, and S. Arun-Kumar. Compositional semantics for real-time distributed computing. *Information and Computation*, 79(3):210–256, 1988.

[9] R. Koymans, J. Vytopil, and W.P. de Roever. Real-time programming and asynchronous message passing. In *Proc. 2nd Annual ACM Symposium on the Principles of Distributed Computing*, pages 187–197, 1983.

[10] N. Lynch and F. Vaandrager. Forward and backward simulations for timing-based systems. In *Real-Time Theory in Practice*, LNCS 600, pages 398–446. Springer-Verlag, 1992.

[11] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, 1992.

[12] D. Peticolas and F.A. Stomp. Compositional reasoning about real-time asynchronous communication with time-outs. In preparation.

[13] A. Salwicki and T. Müldner. On the algorithmic properties of concurrent programs. LNCS 125, pages 169 197. Springer-Verlag, 1981.

[14] F.B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In *Real-Time Theory in Practice*, LNCS 600, pages 618 639. Springer-Verlag, 1992.

[15] A. Shaw. Reasoning about time in higher-level language software. *IEEE TOSE*, SE-15(7):875–899, July 1989.

[16] P. Zhou and J. Hooman. A proof theory for asynchronously communicating real-time systems. In *Proc. Real-Time Systems Symp.*, pages 177 186. IEEE Computer Society Press, 1992.

[17] J. Zwiers. *Compositionality, Concurrency and Partial Correctness.* LNCS 321. Springer-Verlag, 1989.