Of wp and CSP

Carroll Morgan

Abstract

A state-based and an event-based approach to concurrency are linked: the traces, failures and divergences of CSP are expressed as weakest precondition formulae over Action Systems. The result is simpler than is obtained using relations for the state-based view; in particular, divergence is handled easily. Essential use is made of miracles.

Introduction

A typical state-based approach to concurrency is the Action Systems of Back and Kurki-Suonio [1]. A state is shared between a number of actions, each of which is enabled or not depending on that state. The execution of an (enabled) action changes the state, which consequentially changes the set of enabled actions. Unity [2] has the same structure essentially.

A typical event-based approach to concurrency is Communicating Sequential Processes [5]. There the actions (events) have no structure, and affect no state. The behaviour of a process is understood in terms of sequences of those actions (traces) and, for finer distinctions, failures and divergences.

Using state- and event-based approaches together is attractive in practice. There are some aspects of behaviour best described by state (for example, the contents of a buffer); other aspects are best described by explicit sequencing (for example, the exchange of request and confirm messages necessary to set up a communication channel).

Our contribution takes a step towards using Action Systems and CSP together: the three formulae (1) in the concluding section give the traces, failures, and divergences of any action system. The novelty is our use of weakest preconditions rather than relations; the benefit is a simpler formulation.
We present Action Systems first, then CSP very briefly. Finally, we use weakest preconditions to make the link between them.

**Action Systems**

An *action system* is a set of labelled actions and an initialisation; an *action* is a guard and a command; a *guard* is a predicate; and a *command* is a program fragment in Dijkstra’s language of guarded commands [3]. An *initialisation* is a command. Figure 1 is an example of an action system in which the actions have been labelled *Tick* and *Tock*.

\[
\text{initially } n := +1 \\
\text{Tick } \triangleq n = +1 \rightarrow n := -1 \\
\text{Tock } \triangleq n = -1 \rightarrow n := +1
\]

**FIGURE 1.** An action system

Execution of an action system proceeds as follows:

1. First, the initialisation is executed; then
2. Repeatedly an enabled action is selected then executed. An action is *enabled* if its guard is true; it is *executed* by executing its command.

If the repetition in Step 2 fails — because no action is enabled — then the system is said to be *deadlocked*. If a command executed in either step aborts, then the system is said to *diverge*. The behaviour of the action system in Figure 1 is to execute *Tick* and *Tock* alternately and forever, without deadlock or divergence.

**Communicating Sequential Processes**

Seen as a communicating sequential process [5], the behaviour of Figure 1 is a set of traces; they are set out in Figure 2. A *trace* is a sequence of actions.

\{ \langle \rangle, \langle \text{Tick} \rangle, \langle \text{Tick, Tock} \rangle, \langle \text{Tick, Tock, Tick} \rangle, \cdots \} 

**FIGURE 2.** The CSP view: a set of traces

Together with the CSP view comes a notation and a semantics. The traces of Figure 2 are described, by that notation, in Figure 3. Since Figure 1