A Conditional Operator for CCS

Juanito Camilleri  jacl@cl.cam.ac.uk
Computer Laboratory, University of Cambridge,
New Museums Site, Cambridge CB2 3QG, England

Abstract

This paper investigates an extension of Milner’s CCS with a conditional operator called unless [Cam90]. The agent $\mathcal{K} \triangleright E$, pronounced $E$ unless $\mathcal{K}$, behaves as $E$ unless the environment is ready to perform any action in the set $\mathcal{K}$. This dependency on the set of actions the environment is ready to perform goes beyond that encountered in traditional CCS. Its expression is realised by an operational semantics in which transitions carry ready-sets (of the environment) as well as the normal action symbols from CCS. A notion of strong bisimulation is defined on conditional agents via this semantics. It is a congruence and satisfies new equational laws (including a new expansion law) which are shown to be complete for finite agents with the unless operator. The laws are conservative over agents of traditional CCS. The unless operator provides a rudimentary means of expressing bias (or priority) in the behaviour of agents; it is more expressive than the prism operator presented in [CW81] and [Cam90].

1 Introduction

This paper augments Milner’s CCS with a conditional operator called unless. The unless operator can be used to express the behaviour of an agent depending on which actions can take place next in the environment. We will show how the operator provides a syntactic means of expressing a priority structure on actions and moreover, that it can encode a form of multi-way synchronisation. The work presented here can be adopted as a formal semantic foundation for existing programming language constructs that embody the notion of priority between actions [Cam90] [Cam89].

The expressiveness of unless can specify a priority structure on actions that is both dynamic and local. Dynamic in the sense that the priority structure can change as execution proceeds. Local in the sense that the priority structure expressed need not be linear and therefore components of a system may behave independently according to a local bias.

Consider, for example, a reactive system $Sys$ (see Fig.1). An agent $A$ is busy-waiting for the environment to be able to perform $\overline{act}$. In the meantime, another agent $B$ is dormant—i.e., it cannot react while $A$ is busy-waiting. As soon as the environment can perform an $\overline{act}$ action, $A$ is required to stop waiting immediately and it is expected to proceed to handshake with $B$. Following the communication, $A$ is required to stop while $B$ first reacts and then stops. This scenario can be specified very naturally using the unless operator. Suppose the agent $\mathcal{K} \triangleright E$ behaves as $E$ unless the environment is ready to perform any action in $\mathcal{K}$, then the system $Sys$ can then be described as follows:

$$Sys \overset{\text{def}}{=} (A \mid B)\backslash a$$

$$A \overset{\text{def}}{=} \overline{act}.\overline{a}.0 + \{\overline{act}\} \triangleright \tau.A$$

$$B \overset{\text{def}}{=} a.\text{react.0}$$
In Section 5.1, we will show that the behaviour of \textit{Sys} is equivalent to \textit{Spec} defined as follows:

\[
\text{Spec} \triangleq \text{act}\cdot\text{react}\cdot\mathcal{O} + \{\text{act}\} \triangleright \tau \cdot \text{Spec}
\]

In other words, the unless operator allows agent \textit{A} to give precedence to action \textit{act} over the \(\tau\) loop that represents busy-waiting. The semantics of \textit{unless} guarantees that \textit{A} stops busy-waiting as soon as the environment is ready to perform \textit{act}. This example illustrates the expressiveness of the \textit{unless} operator; by relying on fairness and the conventional operators of CCS, one can only guarantee that if the environment can perform \textit{act}, then \textbf{eventually} the agent will stop busy-waiting. In system \textit{Sys} however, we required the response to \textit{act} to be immediate.

Now let us change the scene slightly. Suppose we do not insist that \textit{A} stops busy-waiting as soon as the environment can perform an \textit{act} action. We also require however, agent \textit{B} to \textit{react} immediately after \textit{A} performs \textit{act}—i.e., this should happen without an intermediate handshake between the two agents. In other words, the system comprising \textit{A} and \textit{B} should have the specification \textit{Spec}_0 defined as follows:

\[
\text{Spec}_0 \triangleq \text{act}\cdot\text{react}\cdot\mathcal{O} + \tau \cdot \text{Spec}_0
\]

In CCS, the silent action \(\tau\) plays two roles. First, it can be viewed as an unobservable \textit{skip} operation. Second, it denotes a handshake between two complementary actions. The silent action \(\tau\) in \textit{Spec}_0 adopts the former use.

In this paper, we introduce a new symbol \(\iota\) (iota) distinct from any other action; \(\iota\) is referred to as \textit{idling}. It is intended that \textit{idling} represents a special kind of \textit{skip} operation; namely a skip operation that may be \textit{detected} by the environment. We use the word \textit{detected} and not \textit{observed} to emphasise that \(\iota\) is not intended to have a complement in the traditional sense. Now consider an agent of the form \(\{\iota\} \triangleright E\): this agent behaves as \textit{E} unless \textit{idling} can be detected in the environment.

Sometimes, one needs to suppress or localise an agent’s ability to detect idling. For this purpose a new operator \(\{\}\) is introduced to the language of expressions; the operator is called \textit{masking}. Agent \(\{E\}\) behaves as \textit{E} when \textit{E} cannot detect \textit{idling} in its environment. Masking does not affect an agent’s ability to perform idling. To summarise, we say that an agent \textit{A} detects \textit{idling} in another agent \textit{B}, if \textit{A} is not masked and \textit{idling} occurs as a prefix in one of the sub-agents of \textit{B}.

Now suppose we replace the silent action in \textit{Spec}_0 by an idling action to yield:

\[
\text{Spec}_1 \triangleq \text{act}\cdot\text{react}\cdot\mathcal{O} + \iota \cdot \text{Spec}_1
\]

Intuitively, we expect this specification to be equivalent to \textit{Sys}_1 defined as follows: