CONCURRENT CCS - AN INTRODUCTION

Technical Report No 126

David Smith

April 1991

Concurrent CCS - an Introduction April 1991
David Smith
Hatfield Polytechnic

Abstract

This paper introduces an extension to CCS which models the concept of simultaneity of actions. In order to express simultaneity, we require the semantics of our language to allow agents the facility to proceed concurrently. An example of a safety critical system exemplifies the ideas expressed within this paper and their application.

0. Introduction

CCS[Mil89] is a language used to model communication between concurrent processes or agents. It may be used at various levels of abstraction within the process of specification and design. This paper stems from the observation that in a large class of applications, within the various stages of design, gratuitous and artificial nondeterminism is introduced. This nondeterminism arises, not through the nature of the application, but through the semantics of the design language itself [Bai91]. The introduction of such nondeterminism during design unfortunately leads to implementations that exhibit greater nondeterminism than their specifications. This, at best, may be considered counter intuitive. Equivalence between implementations and their specification is now replaced by the weaker notion of a partial ordering. We address some of these issues here and hope to redress this position by introducing into the specification and design language the facility for actions to occur simultaneously. Simultaneity, as we shall see, naturally arises through the ability of two or more processes to proceed concurrently. Unlike synchronous[Mil83] and timed[To90] CCS, this view of simultaneity does not assume a global clock. Section 1 introduces a case study based upon a level crossing[1] which serves as a vehicle for discussion. Section 2 discusses the problems (given by way of exercises) arising from the design of the level crossing using the standard operators of CCS (in particular composition and restriction). Here we introduce the additional language constructs which are required to adequately capture our intentions. Section 3 formalises these constructs by defining the (operational) semantics of our extended language using transition rules within the context of a labelled transition system. Section 4 presents a brief discussion of equivalence and satisfaction. Here the refinements are "proved" (at a fairly informal level) to satisfy their specifications.

---

1 This level crossing described here is not intended to model level crossings currently in use.

David Smith
Hatfield Polytechnic

1. A Level Crossing

This case study attempts to specify and design within the language of CCS a level crossing system. The level crossing is required to be safe - that is - trains and cars must not be allowed to enter the crossing "at the same time". Safety properties, on their own are easy to specify. As a young woman my sister was informed by our father that in order to avoid an unwanted pregnancy she should sleep with her grandmother. This proved sound advice as a safe method of contraception but rather inappropriate on her wedding night. Likewise, safe crossings can easily be specified by denying any possibility of action. The specification of which in CCS is given by the definition $X_{spec_0} =_{def} 0$ (which, incidently, was the meaning of my fathers advice). Evidently, safety requirements in themselves do not prescribe safe action. Additionally, we require that our system, in some sense, is live.

Liveness may be captured by the *equations*[2] (**)

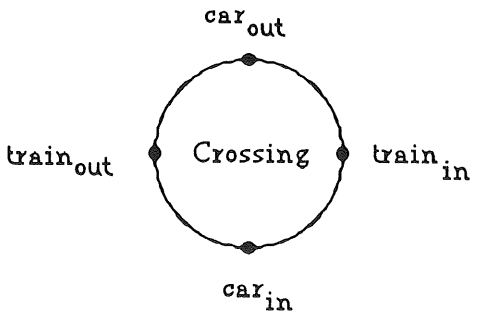
- $Xspec \backslash \{train_{in}, train_{out}\} = Road$
  where $Road =_{def} car_{in}.car_{out}.Road$
- $Xspec \backslash \{car_{in}, car_{out}\} = Track$
  where $Track =_{def} train_{in}.train_{out}.Track$

where X_{spec} is the specification of the system to be designed.

At this stage, as we all know, we wave our magic wand and arrive at a specification which satisfies both the safety and liveness properties, namely

- $X_{spec} =_{def} train_{in}.train_{out}.X_{spec} + car_{in}.car_{out}.X_{spec}$

X_{spec} has the following flow graph



2 For simplicity we shall assume that our crossing may admit at most one car or one train (but, hopefully, not both at the same time) .

David Smith
Hatfield Polytechnic

2. Design

2.1 First Refinement

The design process proceeds as follows:- We may decompose our crossing into two intercommunicating subcomponents - a car control (**Carctrl**) and a train control (**Trainctrl**) each with the following specification.

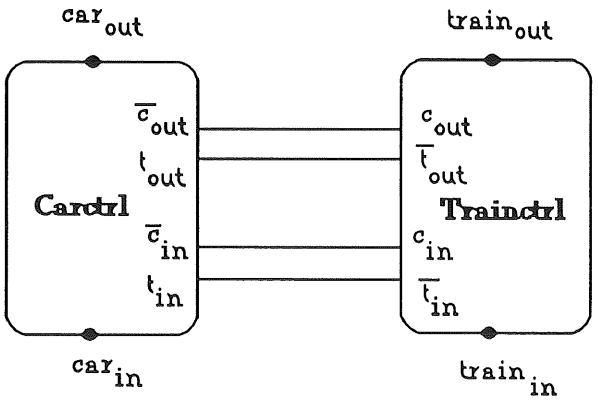
$$\begin{aligned}
&\textbf{Carctrl} &&=_{def} \quad \mathrm{Carctrl}_{go} \\
&\mathrm{Carctrl}_{go} &&=_{def} \quad \mathrm{car}_{in}.\overline{c}_{in}.\mathrm{Sensor}_{car} + t_{in}.\mathrm{Carctrl}_{stop} \\
&\mathrm{Carctrl}_{stop} &&=_{def} \quad t_{out}.\mathrm{Carctrl}_{go} \\
&\mathrm{Sensor}_{car} &&=_{def} \quad \mathrm{car}_{out}.\,\overline{c}_{out}.\mathrm{Carctrl}_{go}
\end{aligned}$$

The intention being:- The car control will only admit at most one car. This only occurs provided a train has not already entered the crossing. The admission of a car will disable the train control. The car will be allowed to leave the crossing, thus enabling the train and car control. If a train had previously entered the crossing then no cars will be admitted until the train has left.

$$\begin{aligned}
&\textbf{Trainctrl} &&=_{def} \quad \mathrm{Trainctrl}_{go} \\
&\mathrm{Trainctrl}_{go} &&=_{def} \quad \mathrm{train}_{in}.\,\overline{t}_{in}.\mathrm{Sensor}_{train} + c_{in}.\mathrm{Trainctrl}_{stop} \\
&\mathrm{Trainctrl}_{stop} &&=_{def} \quad c_{out}.\mathrm{Trainctrl}_{go} \\
&\mathrm{Sensor}_{train} &&=_{def} \quad \mathrm{train}_{out}.\,\overline{t}_{out}.\mathrm{Trainctrl}_{go}
\end{aligned}$$

The train control behaves in an analogous manner.

flow graph for the agent (Carctrl | Trainctrl)$\backslash\{c_{in},c_{out},t_{in},t_{out}\}$



David Smith
Hatfield Polytechnic

Well this all looks hunky dory, however...

- Exercise Show that (Carctrl | Trainctrl)\$\{c_{in},c_{out},t_{in},t_{out}\}$ satisfies the liveness properties expressed in the equations (**)
- Exercise Show that the design leads to an unsafe system (oops... sorry).

We seem to have a problem here. The "crash" arises through the inability of our system to propogate the arrival of either the car or the train "in time". We know that in practice this will not occur - nor will car ferries sail without closing their cargo doors, pilots in airliners shut down "wrong" engines, nuclear reactors in power stations catch fire, underground drivers not stop at terminals...!
As a "solution" to this problem we will permit our crossing only to admit a car (or train) provided the system first disables the Train control (or Car control). This leads to the specification of the subcomponents Carctrl and Trainctrl as

$$\begin{aligned}
&\textbf{Carctrl} &=_{def}\ & \overline{Carctrl}_{go} \\
&Carctrl_{go} &=_{def}\ & \overline{c}_{in}.car_{in}.\overline{Sensor}_{car} + t_{in}.Carctrl_{stop} \\
&Carctrl_{stop} &=_{def}\ & t_{out}.Carctrl_{go} \\
&Sensor_{car} &=_{def}\ & car_{out}.\overline{c}_{out}.Carctrl_{go}
\end{aligned}$$

$$\begin{aligned}
&\textbf{Trainctrl} &=_{def}\ & Trainctrl_{go} \\
&Trainctrl_{go} &=_{def}\ & \overline{t}_{in}.train_{in}.Sensor_{train} + c_{in}.Trainctrl_{stop} \\
&Trainctrl_{stop} &=_{def}\ & c_{out}.Trainctrl_{go} \\
&Sensor_{train} &=_{def}\ & train_{out}.\overline{t}_{out}.Trainctrl_{go}
\end{aligned}$$

However (yes you've guessed)

- Exercise Show that the system is now safe.
- Exercise Show that (Carctrl | Trainctrl)\$\{c_{in},c_{out},t_{in},t_{out}\}$ no longer satisfies the liveness properties (oh s** it, I think I'll build a bridge).

What has gone wrong? As we have seen, the problem has arisen through the inability of CCS to guarantee the immediate propogation of actions. This in turn, arises from the view of concurrency implicit within the language of CCS and related process algebras - namely - the actions resulting from two or more processes proceeding concurrently may not be observed to occur at the same time and by implication (as we are led to believe) concurrency

may be modelled as the arbitrary interleaving of sequential processes. In consequence, this view precludes the facility for actions to occur simultaneously[3]. However, we can argue that the decoupling of the observation (experiment) of a train (or car) entering a crossing from its detection within the crossing is a view that does not conform with our intuitions concerning a crossing's behaviour[4]. The train (or car) being observed to enter a crossing and its detection by the mechanisms of the crossing *should* be modelled as simultaneous actions naturally arising through the ability of (two or more) processes to proceed concurrently. However, we wish to remain true to CCS and capture (any) temporal properties of processes not in terms of an external clock but in terms of the order in which actions may occur. In consequence we consider two or more actions to occur **simultaneously** if

- no restriction is placed on their order of occurrence
- they occur within the context of each other.

In our case study, we require that each collection of actions $\{car_{in}, \overline{c}_{in}\}$ and $\{train_{in}, \overline{t}_{in}\}$ occur simultaneously. It is also natural to consider each collection of actions $\{car_{out}, \overline{c}_{out}\}$ and $\{train_{out}, \overline{t}_{out}\}$ should also occur simultaneously. We hope to show that within the context of design by stepwise refinement (using the operators of composition, restriction and relabelling), simultaneity is exactly the property that we wish to capture.

In order to formally capture simultaneity, we extend the structure of actions by introducing the binary operator •. The intention being that, given any two actions a and b, (a•b) denotes their simultaneous occurrence.

For example, the (composite) action $(car_{in} \bullet \overline{c}_{in})$ expresses our intention that the two actions car_{in} and $\overline{c}_{in}$ are to occur simultaneously. We assume that certain actions such as car_{in} and $\overline{c}_{in}$ are to be designated as atomic or indivisible.

Let us investigate the properties that we require the operators of simultaneity and complementation to possess. Given any two actions a and b, a•b represents their simultaneous occurrence, evidently we would require • to be commutative - this may be expressed equationally as $a \bullet b = b \bullet a$. Similar considerations justifies the requirement that • is also associative. Commutativity and associativity are nice properties for the

3 CCS adopts an ambivalent view of concurrency. Internal communication between two processes may be viewed as the simultaneous occurrence of two complementary actions. However, such simultaneity does not extend to include the occurrence of actions which are observable.

4 We will assume that a train cannot enter a crossing without being detected and that it may not be detected without entering!

operator • to possess. It means that we may write, for example, the action ((b•a)•c) as a•b•c. The action τ, represents the unobserved or internal action, thus τ•a represents the simultaneous occurrence of an internal action with the observed action a. To an observer, this is equivalent to observing the action a. Thus τ•a = a•τ = a. Observe that the simultaneous occurrence of two unobservable actions results in an unobservable action (τ•τ=τ). Complementary actions allow agents the facility to synchronise or communicate, thus the agent (a.p | $\overline{a}$.q) may engage in an internal communication and evolve to (p|q). As such, this is the only property that we require of complementation. We may think of the action a as an input and its complement as the corresponding output with the convention that complementation reverses direction. Thus, for any action a, the complement of $\overline{a}$ is identified with a. The action $\overline{(a \bullet b)}$ represents the output of the simultaneous actions a and b. This is equivalent to the simultaneous output of the action a and the action b. Thus we require complementation to distribute over simultaneity - hence $\overline{(a \bullet b)} = \overline{a} \bullet \overline{b}$. The internal action τ is its own complement ($\overline{\tau} = \tau$).

This view of simultaneity gives rise naturally to the following definition and group properties of the permissable actions. Given a set A (of constant actions), we define $\mathcal{Act}$ = <Act,A $\cup \{\tau\}$,•, $\overline{\ }$ > to be the abelian group with identity τ freely generated by the designated set of constants A and the group operations. The set of constant actions (A) and their complements play a special role in our theory. These will be referred to as **atomic actions.**

Simultaneous actions naturally arise through two or more processes progressing concurrently. We may capture such behaviour by introducing into the language of CCS the operator, || , of concurrent composition[5]. Before we proceed to give the formal semantics of ||, it would be useful to provide an informal description and to contrast || with | (the CCS operator of sequential composition). We illustrate the intended meaning of || by way of an example. This example will be readily recognisable by all those fellow students who are struggling to learn to play the piano.

Suppose the agents Left and Right are defined recursively as

Left	$=_{def}$	g.Left	*A left hand may at any time play the note g.*
Right	$=_{def}$	c.Right	*A right hand may at any time play the note c.*

(Left | Right) models the pianist who may continually play the notes g and c in any order.

5 Although we may capture the ability of processes to proceed concurrently by redefining the semantics of the CCS operator of process composition (|), the new language obtained (CCCS) by introducing the additional operators (|| and •) ensures that CCCS is a conservative extension of CCS.

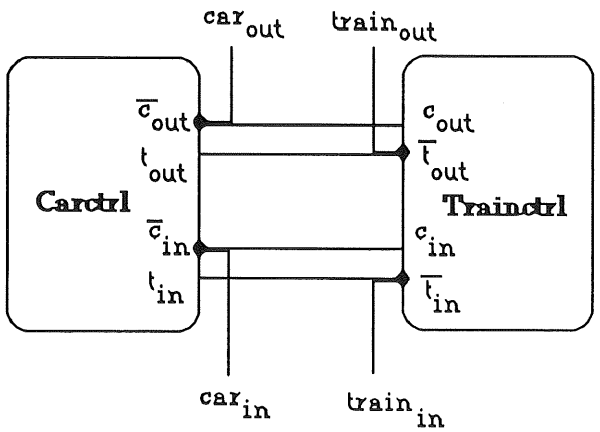
(Left || Right)[6] models the pianist who may may in addition play the notes g and c *at the same time.*
No chords may be heard on the CCS piano, only arpeggios[7].

We will now be able to show that (Carctrl || Trainctrl)\$\{c_{in}, c_{out}, t_{in}, t_{out}\}$ satisfies both the safety and liveness properties of our level crossing system where Carctrl and Trainctrl are now specified as

$$\begin{aligned}
&\textbf{Carctrl} &=_{def}\ & Carctrl_{go}\\
&Carctrl_{go} &=_{def}\ & (car_{in} \bullet \overline{c_{in}}).Sensor_{car} + t_{in}.Carctrl_{stop}\\
&Carctrl_{stop} &=_{def}\ & t_{out}.Carctrl_{go}\\
&Sensor_{car} &=_{def}\ & (car_{out} \bullet \overline{c_{out}}).Carctrl_{go}
\end{aligned}$$

$$\begin{aligned}
&\textbf{Trainctrl} &=_{def}\ & Trainctrl_{go}\\
&Trainctrl_{go} &=_{def}\ & (train_{in} \bullet \overline{t_{in}}).Sensor_{train} + c_{in}.Trainctrl_{stop}\\
&Trainctrl_{stop} &=_{def}\ & c_{out}.Trainctrl_{go}\\
&Sensor_{train} &=_{def}\ & (train_{out} \bullet \overline{t_{out}}).Trainctrl_{go}
\end{aligned}$$

We illustrate this refinement by the following flow graph



We may now, with a clear conscience, proceed with our design as follows:-

---

6 Observe that the "pianist" (Left || Right) satisfies the equation P = a.P + b.P + (a • b).P whereas the "pianist" (Left | Right) satisfies the equation P = a.P + b.P.

7 Strictly, CCS cannot construct a chord from the individual notes.

2.2 Second Refinement

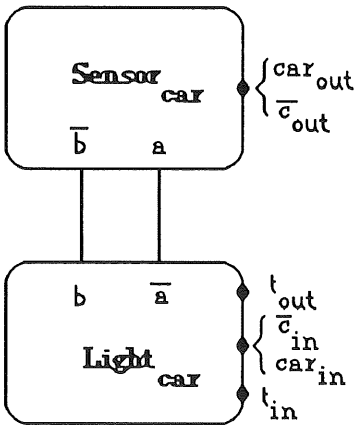
We further decompose Carctrl into two communicating agents - a **Light** and a **Sensor** - as follows:-

$\mathbf{Light}_{car} =_{def} Light_{green}$
$Light_{green} =_{def} (car_{in} \bullet \overline{c}_{in}).\overline{a}.Light_{red} + t_{in}.Light_{flashred}$
$Light_{red} =_{def} b.Light_{green}$
$Light_{flashred} =_{def} t_{out}.Light_{green}$

$\mathbf{Sensor}_{car} =_{def} a.(car_{out} \bullet \overline{c}_{out}).\overline{b}.Sensor_{car}$

The car control consists of two components, a light and a sensor. The purpose of the light is to regulate access to the crossing whereas the sensor detects the departure of cars from the crossing. The sensor is enabled from the light after a car has entered the crossing, it may detect the departure of the car whereupon the train control will be reset and the car light will revert to green. The car light, if green will be switched to flashing red by the arrival of a train or, on the admission of a car, will be switched to red (provided of course no train has previously entered) . The admission of a car will (simultaneously) exclude a train from entering the crossing until after the car has left. On detection of either the train or the car leaving, the light will be switched back to green.

the flow graph for $(Light_{car} \parallel Sensor_{car})\backslash\{a,b\}$ is depicted below



We can now show that $Carctrl = (Light_{car} \parallel Sensor_{car})\backslash\{a,b\}$[8]

8 We can also show that $Carctrl = (Light_{car} \mid Sensor_{car})\backslash\{a,b\}$.

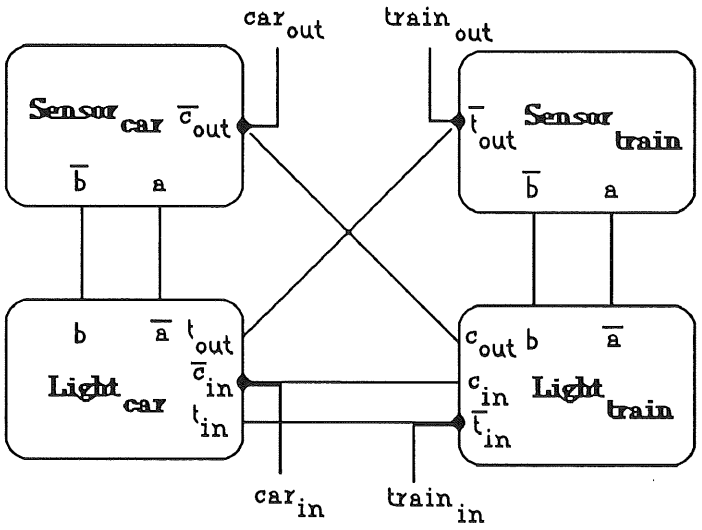
David Smith
Hatfield Polytechnic

A similar decomposition may be applied to Trainctrl. Hence we may conclude that

$$X_{spec} = ((Light_{car} \parallel Sensor_{car}) \backslash \{a,b\} \parallel (Light_{train} \parallel Sensor_{train}) \backslash \{a,b\}) \backslash S$$

where $S = \{c_{in}, c_{out}, t_{in}, t_{out}\}$

This gives the view of our system depicted in the flow graph below.



The following exercise further exemplifies the expressive capabilities of CCCS.

- Exercise Decompose the train control into three intercommunicating agents, a light, a sensor and a bell. The light and sensor are as before. The bell will be turned on immediately a train is detected to enter the crossing and turned off immediately the train leaves the crossing.

- Exercise Draw the corresponding flow graph for the above decomposition.

David Smith
Hatfield Polytechnic

3. Formal Semantics

3.1 Notation

Before we proceed to give the formal semantics of our extended language in terms of a labelled transition system it would be useful to introduce some notation.

- We will write $p \text{ -}^{u}\text{-> } p'$ to mean that the agent p may engage in the collection (multiset) of actions[9] u and evolve to the agent p'. In particular, $p \text{ -}^{\{\}}\text{-> } p'$ denotes the case that may arise when the agent p (unobservably) evolves to the agent p' through some internal communication.
- For each action $s \in Act$, we define the multiset s' as follows:-
  - i) if s is atomic then $s' = \{s\}$
  - ii) $\tau' = \{\}$
  - iii) if $s = (u \bullet v)$ then $s' = u' \oplus v'$ where, in this context, $\oplus$ denotes the operation of multiset union.

Thus for each action s, s' denotes the multiset of all (observable) atomic actions that occur simultaneously.

- We extend the operation of action complementation to multisets (of actions) in a natural way. Thus $\{\}^{c} = \{\}$, $\{a,b\}^{c} = \{\overline{a}, \overline{b}\}$ etc.
- Let u and v denote any two multisets, we define the set $(u \dagger v)$ as

$u \dagger v = \{ s \oplus t \mid s \subseteq u, t \subseteq v, u - s = (v - t)^{c} \}$ where - denotes multiset difference .

The intention expressed here is as follows. Suppose the agents p and q may proceed to engage in the (multiset of) actions u and v respectively. Now the agent $(p \parallel q)$ may proceed concurrently, however this concurrent behaviour may involve the interaction between p and q via complementary actions. $u \dagger v$ is the set of all possible observable actions in which $(p \parallel q)$ may engage (concurrently) in order to evolve to $(p' \parallel q')$. For example, suppose $u = \{a,b\}$ and $v = \{\overline{a}\}$ then $u \dagger v = \{\{b\},\{a,\overline{a},b\}\}$. The multiset $\{b\}$ reflects the possibility that internal communication has occured through the complementary ports a and $\overline{a}$ resulting in the observed action b, whereas the multiset $\{a,\overline{a},b\}$ reflects the alternative possibility, namely all actions have been observed.

The basic language constructs that we shall use to specify and design communicating systems are those of CCS together with the operators $\bullet$ and $\parallel$ introduced earlier. This extension to CCS will be called Concurrent CCS (CCCS).

We introduce the (operational) semantics of the operators (over agents) of CCCS using a labelled transition system (lts). The system that we shall employ here is an extension to the lts of CCS reflecting (possible) concurrent behaviour.

---

9 We will only require the use of this notation in the case that u is empty or it is a non empty multiset of atomic actions.

David Smith
Hatfield Polytechnic

A labelled transition system is a tuple $\langle \mathcal{P}, A, \rightarrow \rangle$ where $\mathcal{P}$ is a set of processes, A is the set of constant actions and $\rightarrow \subseteq \mathcal{P} \times \omega^{A \cup A^c} \times \mathcal{P}$ is the least relation satisfying the set of rules below ($\omega^{A \cup A^c}$ is the set of all finite multisets of atomic actions).

3.2 Inference Rules

We assume that s ranges over actions, p,q range over agents and u,v range over multisets of atomic actions.

- **Action** prefixing

  $s.p \xrightarrow{s'} p$

- **Choice**

  if $p \xrightarrow{u} p'$ then $p + q \xrightarrow{u} p'$

  $q + p \xrightarrow{u} p'$

- **Constants**

  if $p \xrightarrow{u} p'$ and $q =_{def} p$, then $q \xrightarrow{u} p'$

- **Sequential Composition**

  Given any two agents p and q, we define (p | q), the sequential composition of p and q by case analysis as follows

  i) each component of the agent (p | q) may (if possible) proceed independently and interact with its environment.

  ii) each agent may interact with the other independently of the environment resulting in an internal communication

  This is formalised by the following transition rules.

  case i) if $p \xrightarrow{u} p'$ then

  $p \mid q \xrightarrow{u} p' \mid q$

  $q \mid p \xrightarrow{u} q \mid p'$

  case ii) suppose $p \xrightarrow{u} p'$ and $q \xrightarrow{u^c} q'$ then

  $p \mid q \xrightarrow{\{\}} p' \mid q'$

- **Concurrent Composition**

  Given any two agents p and q, we define (p || q), the concurrent composition of p and q by case analysis as follows

  i) as in the case for sequential composition, each component of the agent (p || q) may (if possible) proceed independently and interact with its environment.

  ii) additionally, each agent may proceed concurrently by interacting with its environment and / or with each other.

This is formalised by the following transition rules.

case i) if $p \xrightarrow{u} p'$ then

$p \parallel q \xrightarrow{u} p' \parallel q$

$q \parallel p \xrightarrow{u} q \parallel p'$

case ii) suppose $p \xrightarrow{u} p'$ and $q \xrightarrow{v} q'$ then

$p \parallel q \xrightarrow{r} p' \parallel q'$ for every $r \in (u \dagger v)$

- **Restriction**

  Let S be any set of actions ($\tau \notin S$) and p be the agent such that $p \xrightarrow{u} p'$ then provided $u \cap S = \{\}$ and $u^c \cap S = \{\}$ then $p\backslash S \xrightarrow{u} p'\backslash S$.

- **Relabelling**

  if $p \xrightarrow{u} p'$ and f is a relabelling function then $p[f] \xrightarrow{f(u)} p'[f]$
  where f(u) is the multiset $\{ f(a) \mid a \in u \}$.

Examples

$(a \bullet b).p \xrightarrow{\{a,b\}} p$

$(a \bullet a \bullet a).p \xrightarrow{\{a,a,a\}} p$

$\tau.p \xrightarrow{\{\}} p$

$(a \bullet b).p + a.q \xrightarrow{\{a,b\}} p$

$(a \bullet b).p + a.q \xrightarrow{\{a\}} q$

Suppose $Clock =_{def} (tick \bullet tock).Clock + cough.0$

$Clock \xrightarrow{\{tick,\ tock\}} Clock$

$Clock \xrightarrow{\{\ cough\ \}} 0$

$((a \bullet b).p \parallel (\overline{a} \bullet c).q)) \backslash \{a\} \xrightarrow{\{b,c\}} (p \parallel q)\backslash\{a\}$

$((a \bullet b).p \parallel (c \bullet \overline{b}).0) \backslash \{b\} \xrightarrow{\{a,c\}} (p \parallel 0)\backslash\{b\}$[10]

Suppose $p = (a \bullet b \bullet b).p$, $q = \overline{b}.r$

$(p \parallel (q \parallel q)) \backslash \{b\} \xrightarrow{\{a\}} (p \parallel (r \parallel r)) \backslash \{b\}$ whereas

$(p \mid (q \mid q)) \backslash \{b\}$ and $(p \mid (q \parallel q)) \backslash \{b\}$ are both deadlocked

10 $\{a,c\}$ is the only action in which the agent $((a \bullet b).p \parallel (c \bullet \overline{b}).0) \backslash \{b\}$ may engage.

4. Equivalence and Satisfaction

There are several notions of equivalence prevalent within the literature. Such differering views are generally based upon differing views of non determinism. However, whichever view one adopts, it is generally agreed that agents (descriptions) are taken as equivalent if they, in some sense, exhibit the same behaviour - such behaviour is defined in terms of the actions in which the agents may engage. Certain schools of thought found their theory on the idea of a bisimulation [Mil89], [BW90]. An alternative view is based on may and must testing preorders [Hen88]. The idea of an implementation satisfying its specification is, in general, less well agreed although equivalence is normally taken as a sufficient condition for satisfaction[11]. It is not the purpose here to discuss these differing views of equivalence and satisfaction - however the claims that the designs (above) lead to equivalent systems and hence each refinement satisfies its specification requires some justification. Such justification may be presented in terms of the transition graphs[12] given below.

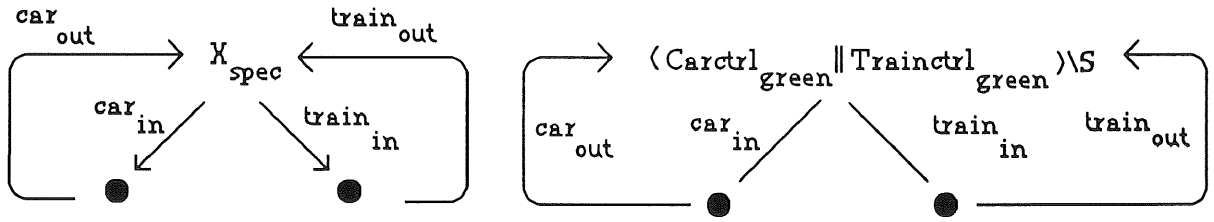


Fig i)

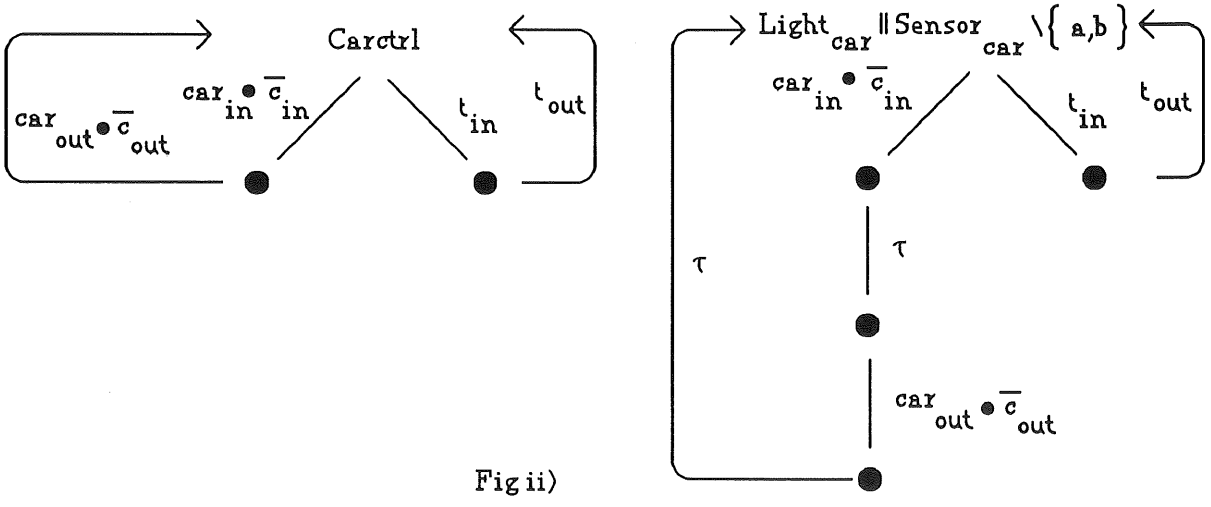


Fig ii)

11 Provided, of course, that agents do not engage in divergent behaviour.

12 Such graphs depict the possible action sequences of a given agent.

Figure i) depicts the transition graphs of (Carctrl || Trainctrl)\$\{c_{in},c_{out},t_{in},t_{out}\}$ and its specification X_{spec}.
Figure ii) depicts the transition graphs of ($Light_{car}$ || $Sensor_{car}$)\$\{a,b\}$ and its specification Carctrl.
The inspection of such graphs is sufficient to guarantee that each agent would be identified with its specification under all prevalent views of equivalence[13].

5 Conclusion

What have we gained? Any attempt to design the crossing within the syntax of pure CCS leads to either one of two alternatives. As we have seen, either the crossing becomes unsafe or it no longer exhibits liveness properties. This arises by decoupling the observation (experiment) of a train (or car) entering a crossing from its detection within the system. Such a view does not conform with our intuitions concerning a crossing's behaviour. The train (or car) being observed to enter a crossing and its detection by the mechanisms of the crossing are naturally modelled as simultaneous actions. The language of CCS and related algebras do not model true concurrency. A major tenet of their formal semantics is that concurrency may be explained in terms of nondeterminism. A parallel process is equivalent to a nondeterministic one obtained by interleaving the actions of its constituent subprocesses[14]. Philosophic justifications for such a view blur the distinction between the ability of concurrent systems to proceed in parallel and the observation of such behaviour. In the view of the author, the extensions to CCS proposed in this paper have considerable applications in many aspects of concurrent system design.

6 References

[Bai91] Baillie E.J., A CCS Case Study: a safety critical system, Software Engineering Journal, July 1991.

[BW90] Baeten J.C.M., Weijland W.P., Process Algebra, Cambridge Tracts in Theoretical Computer Science.

[Hen88] Hennessy, M.C., Algebraic Theory of Processes, MIT Press, 1988

[Mil83] Milner, A.J.R.G., Calculi for Synchrony and Asynchrony, Journal of Theoretical Computer Science, Vol 25, pp267-310, 1983

[Mil89] Milner, A.J.R.G., Communication and Concurrency, Prentice Hall, 1989

[To90] Tofts C., Timing concurrent processes, Technical Report, Department of Computer Science, University of Edinbugh, 1989.

13 Provided, of course we agree to ignore the internal τ actions.

14 Roll over Beethoven.