Mobility in the world alphabetised parallel

Bill Roscoe

Attempting to be true to the spirit of "Open Problems in Concurrency"!

Introduction

- This talk presents questions, some solutions and some challenges, building on and sometimes changing the presentation in Section 20.3 of “Understanding Concurrent Systems (UCS)”.
- Everyone here knows something about CCS, CSP and the π-calculus.
- π-calculus is clearly built on top of CCS. Of course one can explain this in terms of who did what, but it’s interesting to ask what happens when we try to do the same to CSP.
- We will find that the most important difference is in the choices of parallel operator in CCS and CSP.

Background: CCS and CSP

Milner and Hoare developed their process calculi more-or-less simultaneously in the late 1970's. They have a lot of similarities, but we are going to concentrate on the differences!

- Different presentation of choice: $\{+, \tau\}$ versus $\{\Box, \sqcap\}$.
- This had a significant effect on the choice of mathematical theory, leading to operational versus behavioural models.
- CSP has various operators that allow one process to hand control on to a following one: sequential composition, interrupt, throw.
- Factorisation of natural point-to-point parallel with hiding:

Point-to-point parallel with hiding

- CCS: Dual communication model $\alpha, \overline{\alpha}$, parallel $\mid$ means option of α, $\overline{\alpha}$ synchronising as τ or happening independently. This independence can be removed using restriction $\setminus \alpha$.
- CSP: No duals: events synchronise with themselves. Parallel ${}_A\|_B$ synchronises events in $A \cap B$ without turning them into τs. Therefore multi-way parallel is naturally allowed, but we need an extra operator to turn events into τs. $\setminus A$ turns events in A into τs.
- Thus $(P \mid Q) \setminus \alpha, \overline{\alpha}$ and $(P \;{}_{\{a,b\}}\|_{\{a,c\}}\; Q) \setminus \{a\}$ look very similar and mean more or less the same overall, but conceal very different factorisations of this meaning.

CCS versus CSP

One cannot express either process algebra in the other without a fudge factor:

- Synchronisation without hiding, and hence multi-way synchronisation, cannot be achieved in CCS.
- The way that τ resolves $+$ cannot be modelled in CSP, though...
- For each CCS term P there is a CSP translation P' such that $P' \setminus \{Tau\}$ (Tau a special visible event) is strongly bisimilar.

CCS parallel in CSP

$$P \mid Q = (P[\![IP]\!] \underset{\Sigma_1}{\|} Q[\![IDP]\!]) \setminus \Sigma_1$$

where $\Sigma_0 = \{n, \overline{n} \mid n \in L\} \quad \Sigma_1 = \{x' \mid x \in \Sigma_0\}$

Double renamings:

$$IP : x \mapsto x, x' \qquad IDP : x \mapsto x, \overline{x}'$$

The combination of one-to-many renaming and parallel restriction is a powerful tool for expressing “exotic” ideas in CSP.

From CCS to π-calculus

- Turn events into two parts: the channel being used and the (channel) name begin communicated.
- Sometimes restriction placed on recursion, namely using only replication. Does not fit well into the land of CSP!
- Sometimes restriction placed on the use of $+$ to a sub-set of explicitly guarded terms. Fits very well into the land of CSP!
- When a channel name is communicated, there's no need to change any alphabets because there aren't any.
- The interesting bits are the calculus of names and scope extrusion, and how these affect theories such as bisimulation.

π-calculus in CSP

- I showed in an earlier paper[a] how the π-calculus (with recursion and guarded $+$) can be translated into CSP, with options for handling fresh names including nondeterministic choice (over available names) and ensuring uniform order of fresh names down a trace.
- To avoid some conundrums with naming it is better to study this with CSP models that include channel names only in failure/acceptance sets.
- The translation gives a compositional semantics for π-calculus over any such CSP model.
- The fact that CSP models consider linear rather than branching behaviour means that many of the complications of π-calculus semantics disappear, though this needs more research!

[a] CSP is expressive enough for π

Mobility into CSP?

- Is it necessary?
  - Not to show how to create a calculus of mobile concurrency, because we already have one.
  - Not to add to the expressive power of CSP, because we can already express π-calculus in it.
  - But there have been demands from people using CSP who want to include mobile ideas. (See occam-pi, for example.)
  - If we can use FDR on the result, at least some of the time.
- And of course it is interesting to see how the ideas of mobility combine with different process algebras.
- Note the lack of a sense of urgency!

Research programme

- Build appropriate ideas of mobility into CSP in ways that add to the language as far as possible.
- Don't seek pared-down elegance at the expense of usefulness.
- Remain true to the CSP model and style of semantics.
- Make everything mappable to FDR if possible: either by translation to ordinary CSP or by straightforward adaptation of the tool.

Effects of FDR on the development of CSP

- FDR 1991, FDR2 1996, FDR3 2013
- Emphasised its use as a strongly-typed Haskell-like functional language with process algebra operators, because we need the power to describe the complex systems FDR can analyse.
- Focussed attention on static networks with parallel/hiding compositions at the outermost layers – the class that FDR optimises – to the extent that complex components are typically factored as parallel compositions for efficiency within FDR.
- In many ways the opposite of π-calculus.

Types

- CSP has evolved to be a strongly typed declarative language akin to Haskell.
- This is greatly reinforced by FDR3, which integrates a type checker.
- Therefore we include type for communicable channels port of T for any communicable type T. (Different from UCS, where there was just undifferentiated port.)

  So port of port of Int is an example.

  New channels are thus declared with types, whether these include ports or not.
- This means that no channel can be communicated down itself.

Alphabets

- Hoare's treatment of CSP gave every process its own alphabet, like a type, though alphabets were only really used for parallel: $P \parallel Q$ means that P has to agree on all in αP, and Q all in αQ. Elegant but much more programming overhead.
- My books have assumed that all processes are composable with each other, but introduce explicit alphabets in parallel: $P \;{}_X\!\parallel_Y\; Q$ treats X and Y as αP and αQ, $P \underset{X}{\parallel} Q$ makes X the interface, allowing P and Q to communicate freely outside.
- In any mobile CSP, alphabets are going to have to change dynamically:
  - Which of the above approaches can handle this?
  - Does interface parallel make sense?
  - How about the factorisation of point-to-point parallel?

Pass the port

- In π-calculus the only thing a process needs to use a channel is knowledge of its name. The parallel operator handles this naturally and without alteration.
- In CSP, we are going to have to change a process's alphabet to enable it to use any channel that was not initially in its alphabet.
- Desirable to have linguistic flexibility to handle
  - Inputting a port and adding it to our alphabet: $c?p+$
  - Outputting a port and subtracting it from our alphabet: $c!p-$
  - Doing these things without changing our alphabet: $c?p$, $c!p$.
  - Doing multiple things in a single action: $c?x!p-?q+$.
  - To use such communicated channels for multi-way synchronisation, hiding etc.

Double think

- If an entity outside the process needs to know its changing alphabet, the difference between $c.p$, $c.p+$ and $c.p-$ needs to be visible to it.
- But clearly $c.p+$ in one process needs to be able to synchronise with $c.p-$ in another.
- Probably the correct solution to this is to have the processes perform events that are explicitly decorated, but normalise these before synchronisation.

Eliminating alphabets algebraically

It was observed many years ago that various identities such as the following hold:

$$P \;{}_{X}\|_{Y}\; Q = (P \underset{X}{\|} RUN_Y) \underset{\Sigma}{\|} (Q \underset{Y}{\|} RUN_X)$$

Completely synchronising two processes which allow any action outside the natural alphabets.

$$P \;{}_{X}\|_{Y}\; Q = ((P \underset{X}{\|} RUN_\Sigma) \underset{\Sigma}{\|} (Q \underset{Y}{\|} RUN_\Sigma)) \;{}_{\Sigma}\|_{\Sigma-(X\cup Y)}\; STOP$$

Similar, but restricting to $X \cup Y$ via an extra parallel composition.

We can let a process communicate in events outside its natural alphabet provided it always accepts them....

This inspires implementations of mobile parallel in which these RUN processes are elaborated so that they always contribute just the right extra communications.

Modelling channel mobility in multi-way fixed-alphabet parallel

It is possible to achieve this sort of effect within standard CSP where there are no dynamic alphabets: to model the dynamic-alphabet network $\hat{\|}_{i=1}^{n}(P_i, A_i)$, take the composition

$$Reg \underset{AReg}{\|} (\|_{i=1}^{n} (\hat{P}_i, \hat{A}_i))[\![\hat{R}]\!]$$

- $\hat{A}_i = A_i \cup M$, where M are all ports.
- $\hat{P}_i = (P_i \parallel Q_i)[\![R_i]\!]$
- Q_i keeps track of P_i's dynamic alphabet, and permits all mobile actions outside it.
- The renaming R_i drops decorations to allow synchronisation

Rest of the plumbing

- $\hat{R}$ puts the $+$, $-$ decorations back on for Reg to choose between.
- Reg also prevents the whole composition performing actions that are in no P_i's alphabet.

Most of this translation was described in detail Chapter 20 of UCS, though that was restricted to closed world parallel (i.e. union of A_i invariant), so there was no need for $\hat{R}$ or Reg.

Example

- In UCS I described a telephone system in which calls are made by passing mobile channels, through an exchange network, from one phone to another.
- Fully implemented in CSP$_M$, supplying a library for interpreting a slightly limited mobile CSP. Properties checkable on FDR in networks of a few each of phones and exchange nodes.
- I've used this example twice as part of the assessment for my annual course on the advanced use of CSP/FDR: getting students to add features to the phone system.

Sample CSP$_M$

```
Ringing(x,c) =
   ringing.x -> Ringing(x,c)
[] lift.x -> mc.c.Pickup!ch(x).Plus ->
      (mc.ch(x).Confirm!Plus -> Incall(x,c)
      [] mc.ch(x).Hungup!Plus -> mc.c.Cancel.Minus -> CallOver(x))
[] busy!x?e -> Ringing(x,c)
[] mc.c.Cancel.Minus -> Phone(x)
```

NB: The coding used here assumes that all communications on mobile channels have `Plus/Minus` decorations.

Operational theory

$$\frac{P \xrightarrow{\tau} P'}{P \;{}_X\|_Y\; Q \xrightarrow{\tau} P' \;{}_X\|_Y\; Q} \qquad \frac{Q \xrightarrow{\tau} Q'}{P \;{}_X\|_Y\; Q \xrightarrow{\tau} P \;{}_X\|_Y\; Q'}$$

$$\frac{P \xrightarrow{a} P'}{P \;{}_X\|_Y\; Q \xrightarrow{\xi_1(a,X,Y)} P' \;{}_{\Xi(a,X)}\|_Y\; Q} \quad (\psi(a) \in X - Y)$$

$$\frac{Q \xrightarrow{a} Q'}{P \;{}_X\|_Y\; Q \xrightarrow{\xi_1(a,Y,X)} P \;{}_X\|_{\Xi(a,Y)}\; Q} \quad (\psi(a) \in Y - X)$$

$$\frac{P \xrightarrow{a_P} P' \wedge Q \xrightarrow{a_Q} Q' \wedge \psi(a_P) = \psi(a_Q)}{P \;{}_X\|_Y\; Q \xrightarrow{\xi_2(a_P,a_Q,X,Y)} P' \;{}_{\Xi(a_P,X)}\|_{\Xi(a_Q,Y)}\; Q} \quad (\psi(a) \in X \cap Y)$$

Handling plusses and minusses

- $\psi(a)$ removes alphabet decorations $+, -$
- $\Xi(a, X)$ adjusts X relative to the action a.
- $\xi(a, X, Y)$ adjusts the decorations on event a relative to its effect on the combination of alphabets $X \cup Y$ when performed by only the first argument of ${}_X\|_Y$.
- $\xi_2(a_P, a_Q, X, Y)$, where $\psi(a_P) = \psi(a_Q)$ adjusts the decorations on the event $\psi(a_P)$ relative to its effects on $X \cup Y$.

Definitely a CSP-like CLASS of operators, as are indexed versions.

Intrinsic alphabets

- It is tempting to move back to a process having its alphabet intrinsically rather than specified in the parallel operator.
- Would make little difference to the description of the parallel operator.
- Would need to maintain alphabets of all other constructs....
- What happens with cases like $(c.p+ \to P) \setminus \{| \ c \ |\}$?
- Wild suggestion.... see below.

Other operators

- We need to be careful with all other CSP operators parametrised by events, so as to avoid non-intuitive behaviour.
- Not clear what to do about interface parallel $P \underset{X}{\|} Q$, particularly how to handle the case where P's and Q's natural alphabets intersect outside X.

  So it may be necessary to restrict this to cases where the interface is static.

Hiding

- No obivous way to get the hidden set in $P \setminus X$ to vary dynamically which we might well want in hiding the internal communcations of parallel compositions.
- Presents no problems with closed-world mobile parallel: just hide at the outside.
- To solve this problem in general it may well be best to have the option of a parallel operator which, like CCS parallel, hides synchronised events.

Further operators

- Remaming: probably has to be restricted so that on events involving ports it is by injective substitution of ports.
- Link parallel: $P[a \leftrightarrow b]Q$
- Need to be careful about the alphabet of Q in contexts like

$$(P * Q)\ {}_X\|_Y\ R$$

for $* \in \{;\ ,\triangle,\Theta_a\}$.

Wild suggestion

In process algebra semantics we generally assume that processes only perform the actions that the natural interpretation says they can.

But we have used the mapping $P \Rightarrow \hat{P}$ in implementing mobility: adding all mobile communications that P cannot block (i.e. outside its current alphabet).

Might we use $\hat{P}$ (interpreted in an LTS or CSP model such as traces) as the basic semantics of P?

Wild suggestion

- Works smoothly for parallel (as you might expect), but otherwise messy.
- Do we worry about not distinguishing between a process that has an action $a \in \alpha P$ and always allows it, from one such that $a \notin \alpha P$? I think I do!
- Renaming problematic.
- I dangle this as an interesting suggestion....

Dynamic networks and scope extrusion

Dynamic networks are natural in CSP, but traditionally have been constructed by re-using channel names rather than inventing lots of fresh ones:

$$B^\infty = \mathit{left}?x \to (B^\infty[\mathit{right} \leftrightarrow \mathit{left}]\mathit{right}!x \to \mathit{COPY})$$

However, as the CSP semantics for π-calculus shows, we are free to have constructs such as $\nu n : T.P$, with T a type of ports.

Semantic issues might necessitate linguistic restrictions so that channel-only failure/acceptance sets are valid.

Conclusions

- It seems to be possible to add mobility into a basic CSP language of prefix, choice, alphabetised parallel and recursion.
- Seems natural and most useful to extend the CSP$_M$ type system.
- But we need to be a bit careful not to get counter-intuitive results with hiding and renaming – and therefore perhaps ought to have the option of re-combining parallel and hiding.
- And combining dynamic alphabets with other constructs may not work well.

Verification on FDR

- Because of the nature of FDR will tend to work better on fixed-sized networks with mobile channels – the closed world of UCS.
- The current project on lazy compilation will enable a wider scope, still for finite-state systems.
- At present will only work by translation to standard CSP, but a later version FDR3 will offer support for any CSP-like language.