my DCS file

SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

COMPUTING DIVISION

D I S T R I B U T E D C O M P U T I N G N O T E 5 2 4

VISITS
Notes on a visit to R Milner and G Plotkin
18 November 1981

issued by
D A Duce

25 November 1981

DISTRIBUTION:
R W Witty
F Chambers (Logica Ltd)
D A Duce
Miss G P Jones
Investigators/Plotkin and Milner

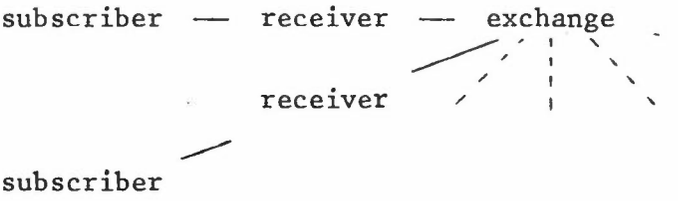
STL COOPERATIVE GRANT

Robin Milner started by describing the STL cooperative grant.

This work is concerned with the application of CCS to switching systems. STL are providing a number of non-trivial case studies. The project started in October 1980 and funds one RA, Mike Shields.

STL use the CCITT descriptive language Systems Description Language (SDL) as a design aid. Mike has discovered that this maps fairly directly to CCS.

As an example consider a telephone system:

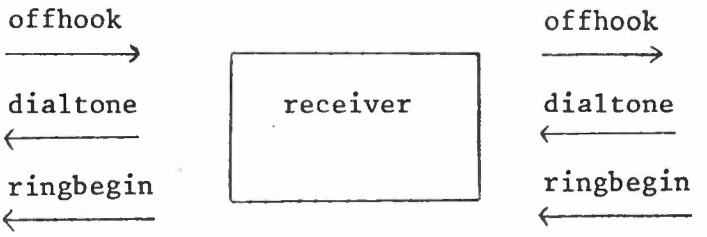


Consider a description of how the exchange + receivers looks to subscribers.

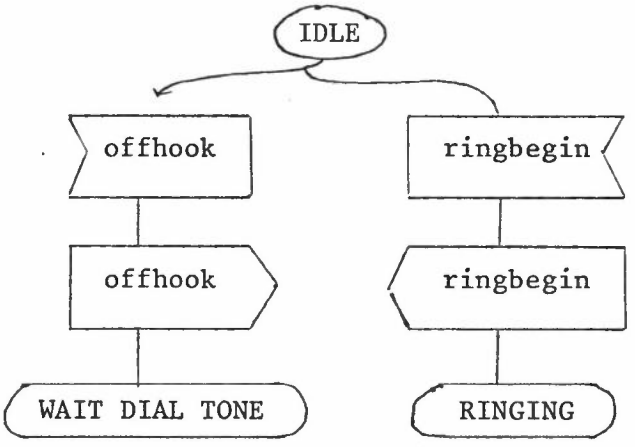
Think of the receiver as a black box, transmitting messages transparently but interpreting some messages. This can be described by a state transition diagram.

An SDL description consists of a number of boxes representing state linked with other boxes representing the transmission and reception of messages.

The receiver can be described by the following signals (simplified!)

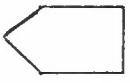


This could be represented in SDL by the diagram:



where

represents an input message and:



an output message.

In CCS, IDLE is a behaviour expression consisting of two possibilities:

$$\text{IDLE} \Leftarrow (\quad) + (\quad)$$

The '+' operator means sum of possibilities; precise sumantics can be given.

The other notation necessary to complete the description is

$$\alpha.B$$

where α is a label and B an expression standing for a behaviour. Inputs are represented by α, outputs by $\bar{\alpha}$. To distinguish signals on the subscriber and exchange side of the receiver, the latter are primed.

The description of the above model then becomes:

$$\text{IDLE} \Leftarrow (\text{offhook} \,.\, \overline{\text{offhook}}' \,.\, \text{WAITDIALTONE}) + (\text{ringbegin}' \,.\, \overline{\text{ringbegin}} \,.\, \text{RINGING})$$

There is a mismatch between SDL and CCS in the way message boxes meet:

buffering between transmitter and receiver is implicit in SDL, but is not a primitive in CCS.

SDL is a purely descriptive aid. The manual contains about 1 page on the meaning of the language and 5 on the proportions of boxes!

Mike is interacting with SDL on the semantics of the language.

Designers of switching systems are seeking formalisms which are amenable to verification etc but no one knows what these should be.

Mike is also looking at the relationship between Petri nets (also used by STL) and SDL/CCS.

DCS GRANT

Two RA's have now been recruited for the new DCS grant, starting early in 1982. One RA, Colin Sterling has a PhD in modal and temporal logics. In his view most of the work done in the USA on proof techniques for parallel programs has been done in ignorance of recent work in temporal logic and is based on inadequate formalisms. A deep understanding of logics is a necessary prerequisite of the next phase of the work on proof techniques as it is not yet clear what logic should underpin the theory as it is not clear what assertions need to be made. Gordon Plotkin then described their recent work in operational semantics.

This is based on the abstract machine approach to operational semantics (Landin's SECD machine, VDL etc) but with the important difference that configurations are to be kept simple.

To execute a command c, one starts with an initial state σ; combining these one obtains an initial configuration:

$$\gamma_o = \langle c, \sigma \rangle$$

The execution proceeds through discrete stages, represented by further configurations, which have either the form:

$$\gamma' = \langle c', \sigma' \rangle$$

where c' represents the remaining computation, or

$$\gamma' = \sigma'$$

where σ' is the resulting state at termination of c.

An execution is formalised as a sequence:

$$\gamma \rightarrow \gamma' \rightarrow \ldots.$$

of configurations. The arrows represent transitions from one stage to another.

Consider an example, the semantics of $c_0;c_1$ ie execution of c_0 followed by c_1.

Execution of c_0 begins from an initial state σ. Consider the transition $<c_0,\sigma> \rightarrow \gamma'$. There are two cases according to the form of γ'. First suppose it is $<c^1,\sigma'>$. Then the initial state of the second stage of the execution of $c_0;c_1$ must be σ'. The rest of the execution of $c_0;c_1$ is the rest of the execution of c_0 (represented as c_0') followed by c_1. Thus γ' can be taken as

$$\gamma' = <c_0';c_1,\sigma'>$$

and the following rule has been informally justified:

$$\frac{<c_0,\sigma> \rightarrow >c_0',\sigma'>}{<c_0;c_1,\sigma'> \rightarrow <c_0';c_1,\sigma'>} \qquad \begin{matrix}(a)\\(b)\end{matrix}$$

which is a rule of inference that if (a) is known then so is (b). The second case is that execution of c_0 terminates in state σ', in which case:

$$\frac{<c_0,\sigma> \rightarrow \sigma'}{<c_0;c_1,\sigma> \rightarrow <c_1,\sigma'>}$$

For example:

$$<z:=x;\ \ x:=y;\ \ y:=z,\ \{x=1,\ y=2,\ z=3\}>$$

assuming assignment is an atomic command, ie

$$<z:=x,\ \{x=1,\ y=2,\ z=3\}> \rightarrow \{x=1,\ y=2,\ z=1\}$$

by case 2 above:

$$<z:=x;\ (x:=y;\ y:=z),\ \{x=1,\ y=2,\ z=3\}>$$

$$\rightarrow\ <x:=y;\ y:=z,\ \{x=1,\ y=2,\ z=1\}>$$

$$\rightarrow <y:=z,\ \{x=2,\ y=2,\ z=1\}>$$

$$\rightarrow \{x=2,\ y=1,\ z=1\}$$

An atomic command always demolishes itself on execution. In general assignment is not atomic but involves some expression evaluation, eg:

$$\frac{\langle e,\sigma\rangle \rightarrow \langle e' \sigma'\rangle}{\langle z:=e,\sigma\rangle \rightarrow \langle z:=e',\sigma'\rangle}$$

Note that the semantics of ';' are invariant, it is the semantics of assignment that become complex!

As a second example consider a while loop. Assume booleans evaluate to time (tt) or false (ff) with no side effects. Then:

$$\frac{\langle b,\sigma\rangle \rightarrow tt \mid ff}{\langle \underline{\text{while}}\ b\ \underline{\text{do}}\ c,\sigma\rangle \rightarrow \langle c;\ \underline{\text{while}}\ b\ \underline{\text{do}}\ c,\sigma\rangle \mid \sigma}$$

To handle say jumps requires the addition of a configuration abort which leads to the rule:

$$\frac{\langle c_o,\sigma\rangle \rightarrow \underline{\text{abort}}}{\langle c_o;c_1,\sigma\rangle \rightarrow \underline{\text{abort}}}$$

These ideas can handle sequential languages nicely. They can also be extended to non-deterministic languages.

Consider communication:

$$\langle p!x,\ \{x=5\}\rangle \rightarrow \{x=5\}$$

where p!x means send x to p. This operation does not affect the state σ. Without knowing p, one cannot see how to change the state, so "outputting 5 to p" is written as:

$$\langle p!x,\ \{x=5\}\rangle \xrightarrow{p!5} \{x=5\}$$

labelling the transition with a description of the interaction of p!x with its environment.

To give a semantics for Hoare's CSP requires the addition of status abort and fail.

Syntax:

Q!3	send 3 to Q agent unknown
P,Q!3	P sends 3 to Q
P?4	P gets 4
P,Q?4	

Semicolon remains safe:

$$\frac{<c_0,\sigma> \xrightarrow{a} <c_0',\sigma'>|\sigma' \mid \underline{\text{abort}} \mid \underline{\text{fail}}}{<c_0;\sigma c_1,> \xrightarrow{a} <c_0';c_1,\sigma'>|<c_1,\sigma'>| \underline{\text{abort}} \mid \underline{\text{fail}}}$$

Consider the parbegin operator:

$$<c_0||c_1,\sigma>$$

$$|| = ;U;^{-1}$$

we have:

$$\frac{<c_0,\sigma> \rightarrow <c_0',\sigma'>|\sigma'}{<c_0||c_1,\sigma> \rightarrow <c_0'||c_1,\sigma'>\Big|^{\sigma'}_{<c_1',\sigma'>}}$$

Introducing the configuration fail into the discussion leads to a delicate argument about what the failure of one component should mean. If c_0 fails, should c_1 continue or fail also.

If labelled transitions are introduced (ie P sends a message and Q receives it)

$$\frac{<c_0,\sigma> \xrightarrow{P,Q!M} <c_0',\sigma'>|\sigma' \quad <c_1,\sigma> \xrightarrow{P,Q?M} <c_1',\sigma''>|\sigma''}{<c_0||c_1,\sigma> \xrightarrow{\varepsilon} <c_0'||c_1',\sigma''>}$$

Points to note:

1 Sending something does not affect the state. It needs to be proven that $\sigma' = \sigma$.

2 In the infered transition the label is ε because the communication is internal to c_0, c_1.

If c_0 and c_1 swapped values, the final state would be $\sigma' + \sigma''$ and one has to consider carefully what state this is, by considering local stores etc. The constraints that Hoare intuitively places on his language avoids these difficulties (by not allowing output commands in guards).

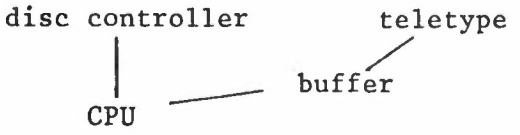
One can attempt to apply these semantics to any language. The more realistic the language the harder it becomes. A student has been looking at the task parts of Ada (primitives and exception raising). Many problems arise because it is not clear what the Ada document means, eg if two tasks each raise an error in the other, can both succeed?

Gordon has produced a paper on the operational semantics of CSP (copy in file).

DISCUSSION

Original goals of the project included integrating Petri nets, Flow Algebras, CSP, actors etc.

Consider:



A system can be viewed as some arbitrary graph, showing where rather than when communications can possibly occur. Flow algebra was limited to saying how such structures could be manipulated algebraically.

Think of a third dimension normal to the above plane, representing time. The question then arises how to deal with the dynamics of such a structure algebraically and integrate this with the flow algebra static description. CCS is the result.

The project has succeeded in relating CCS and CSP, but hasn't really succeeded in elucidating the link with Petri nets. However, parts of CCS have been translated into Petri nets, sumantics given and proved correct.

Part of the problem with elucidating links to Petri nets and especially Hewitt's actors is that the internal theories of these systems are not well developed. In the actor model for example the linkage structure can evolve through time in a very rich way.

A student is looking at an architecture to implement CCS.

The original version of CCS was asynchronous and couldn't handle eg time outs, though real time might have been handled by adding clocks as fictious agents.

A synchronous version, SCCS, has been described which makes the assumption that there is a global time. This gives a very nice albegra from which the asynchronous algebra can be derived.

The problem of fair computation is now being considered. CCS has operations '+' where B+C means do B or C (may be non-deterministic) and '|' where B|C means do B in parallel with C. In the expression (B|C)|A it is not determined which of B and C communicates with A if A only wishes to communicate with one of them. Non-determinism is thus not represented by a single operator.

In the synchronous calculus '|' is replaced by 'x' which has no element of non determinism. In B|C non determinism arises because of indeterminate delays involved. Explicit delays are represented in the synchronous calculus by δ. Thus:

$$B|C = (Bx\delta c + \delta Bxc)$$

Non-determinism is confined to the operator '+' and parallelism to 'x'.

The concepts of action, composition, alternation, scoping etc are represented almost by single constructs. Urgency and fairnéss are not however represented. The expression

$$\delta B = B + 1 : \delta B$$

where 1 is a unit time step admits infinite delays. An operator ϵ satisfying this equation but where the infinite path is ruled out is sought. ϵ is beginning to look consistent and manageable, but this is still an open question. Fairness is generating a lot of interest in many groups.

Hewitt's actors assume that you will get a message sometime. This is sufficient of a fairness assumption that it is intractable.

Robin and Gordon are looking for proof techniques, but these are still a long way away. It is still not clear what logic should underpin proof techniques.

MISCELLANEOUS

Robin, Gordon and Rod Burstale have a 3 year grant from the BP venture unit which enables them to spend 1 year each doing nothing but research. The order in which the year will be taken is as above. Robin started his study leave in October this year.

The only other grant given in computer science to date was to one of Dijkstra's students.