CHAPTER 5

Syntax and Semantics of CCS

5.1 Introduction

We have seen some examples of expressions of CCS, representing both programs and their specifications. We saw that, with the introduction of value-passing, we had to abandon the simple interpretation of behaviour expressions as synchronization trees, but in §4.2 we talked of atomic experiments on behaviour expressions (or on the behaviours for which they stand), and this was developed further in §4.4 on derivations.

We are now ready to present CCS precisely, and to define precisely the atomic actions (and hence the derivations) of every CCS program. On this basis, we proceed in this chapter and in Chapter 7 to develop our central notion, observation equivalence of programs. From this it is a short step to a congruence relation; two programs are observation congruent iff they are observation equivalent (i.e. indistinguishable by observation) in every context. Our proposal is that an observation congruence class is a behaviour, so that CCS is indeed an algebra of behaviours, in which each program stands for its congruence class.

This main development is independent of the notion of ST. STs may now be regarded as a first approximation (not sufficiently abstract) to a model of CCS without value-passing, and in Chapter 6 we show how they may be generalised to CTs (communication trees) to give a first approximation to a model of CCS with value-passing; again, the main development is independent of CTs, which are only discussed to aid understanding. When we eventually define observation equivalence over programs in Chapter 7, it will look just like the corresponding definition in §3.3 over STs, which generalises to CTs in an obvious way. Indeed, we expect to find that two programs are equivalent iff the corresponding CTs are so; in that case CTs, though not technically essential, fit naturally into our picture.

This chapter is devoted to a congruence over programs which we call strong congruence, since it is stronger than the observation congruence studied in Chapter 7. By approaching our proposal in two stages we introduce the properties of behaviour gradually, and with greater insight than if we tackled observation congruence immediately. In fact we even subdivide the first stage in this chapter, approaching strong congruence via an even

stronger relation called direct equivalence.

The CCS language was introduced in the author's "Synthesis of Communicating Behaviour" [Mil 3]. However, the semantic specification by derivations was not given there in detail.

5.2 Syntax

Value expressions E

Value expressions are built from

(i) Variables $x,y,\ldots$

(ii) Constant symbols, and function symbols standing for known total functions over values

using conventional notation. We also allow tuples $(E_1,\ldots,E_n)$ of value expressions. Thus each value expression without variables stands for a uniquely defined value; we shall not worry about the distinction between such expressions and their values.

We shall also avoid details about the types of values and value expressions, though we shall have to mention some syntactic constraints depending on such details (which are standard).

Labels, sorts and relabelling

As in Chapter 2, our labels are $\Lambda = \Delta \cup \bar{\Delta}$, together with τ. We use $\alpha,\beta,\ldots$ to range over Δ, λ over Λ, and $\mu,\nu,\ldots$ to range over $\Lambda \cup \{\tau\}$. A sort L is a subset of Λ; to each behaviour expression B will be assigned a sort L(B). †

A relabelling $S : L \to M$ between sorts L and M is as in §2.2. However, some positive labels α will be used to bind (tuples of) variables, and then $\bar{\alpha}$ will qualify (tuples of) value expressions; we must ensure that S preserves the sign of such labels (i.e. $S(\alpha) \in \Delta$). Moreover, in a complete treatment we should have to assign types to value variables and value expressions, hence also to labels, and to ensure that relabellings respect the types of labels. We will avoid these details; they need care, but would only obscure the more important aspects of semantics which we want to discuss here.

† We shall only meet finite sorts in examples. However, all we need to assume - for technical reasons - is that Λ is never exhausted. Infinite sorts may be of use; see the end of Chapter 6.

Behaviour identifiers b

We presuppose a collection of such identifiers, each having preassigned

(i) an arity n(b) - the number of value parameters.
(ii) a sort L(b).

We assume that the meaning of such identifiers is given, often recursively, by a behaviour expression. For example, in §4.5 we gave meaning to the behaviour identifier p by

$$p(x) = \bar{\alpha}_1(x,f(x)).\alpha_2 x'.\ p(x')$$

where $n(p) = 1$, $L(p) = \{\bar{\alpha}_1,\alpha_2\}$.

Again, a complete treatment would specify not just an arity but a type (i.e. list of parameter types) for each b .

Behaviour expressions B

Behaviour expressions are formed by our six kinds of behaviour operator (§4.1), by parameterising behaviour identifiers, and by conditionals. It's convenient to present the formation rules as a table (see below), giving for each expression B its sort L(B) and its free variable set FV(B).

We should regard the language given by the table as a core language, which we are free to extend by defining derived behaviour operators (the chaining combinator $\frown$ of §4.1 for example) and by alternative syntax for commonly occurring patterns.

In what follows, we shall use

$$B\{E_1/x_1,\ldots,E_n/x_n\}$$

to denote the result of substituting expression E_i for variable x_i $(1 \le i \le n)$ at all its free occurrences within B . Sometimes we shall abbreviate vectors (tuples) of variables and expressions as $\tilde{x}$ and $\tilde{E}$, and write a substitution as

$$B\{\tilde{E}/\tilde{x}\} .$$

(As usual, such substitutions may require change of bound variables, to avoid clashes.)

SYNTAX TABLE FOR BEHAVIOUR EXPRESSIONS

Form	B"	L(B")	FV(B")
Inaction	NIL	$\emptyset$	$\emptyset$
Summation	$B + B'$	$L(B) \cup L(B')$	$FV(B) \cup FV(B')$
Action	$\alpha x_1,\ldots,x_n \,.\, B$	$L(B) \cup \{\alpha\}$	$FV(B) - \{x_1,\ldots,x_n\}$
	$\bar{\alpha} E_1,\ldots,E_n \,.\, B$	$L(B) \cup \{\bar{\alpha}\}$	$FV(B) \cup \bigcup_i FV(E_i)$
	$\tau.B$	$L(B)$	$FV(B)$
Composition	$B \mid B'$	$L(B) \cup L(B')$	$FV(B) \cup FV(B')$
Restriction	$B \setminus \alpha$	$L(B) - \{\alpha,\bar{\alpha}\}$	$FV(B)$
Relabelling	$B[S]$	$S(L(B))$	$FV(B)$
Identifier	$b(E_1,\ldots,E_{n(b)})$	$L(b)$	$\bigcup_i FV(E_i)$
Conditional	if E then B else B'	$L(B) \cup L(B')$	$FV(E) \cup FV(B) \cup FV(B')$

The table shows how B" of sort L(B") may be built from B,B' of sorts L(B),L(B'). Parentheses are to be used to make parsing unambiguous, or to emphasize structure; to avoid excessive use of parentheses we assume the operator precedences

$$\left\{\begin{matrix}\text{Restriction}\\ \text{Relabelling}\end{matrix}\right\} > \text{Action} > \text{Composition} > \text{Summation} .$$

Thus for example

$$B \mid \tau.B'\setminus\alpha + B"[S] \quad \text{means} \quad (B \mid (\tau.(B'\setminus\alpha))) + (B"[S]) .$$

5.3 Semantics by derivations

We proceed to define a binary relation $\xrightarrow{\mu v}$ over behaviour expressions, for each $\mu \in \Lambda \cup \{\tau\}$ and value v (of type appropriate to μ). $B \xrightarrow{\mu v} B'$ may be read "B produces (or can produce) B' under μv"; thus if B, B' are in the relation $\xrightarrow{\mu v}$, a particular atomic action of B - resulting in B' - is indicated.

Referring back to §3.3, we are taking behaviour expressions to be our agents; towards the end of §3.3 we chose STs as agents, and we shall see in the next chapter how to regard CTs as agents.

Note that $\xrightarrow{\tau}$ is a particular case of our relations, since the only value of type appropriate to τ is the 0-tuple.

The relations $\xrightarrow{\mu v}$ are defined by induction on the structure of behaviour expressions. This means that all the atomic actions of a compound expression can be inferred from the atomic actions of its component(s).

Such a relation, though not indexed as here by μv, probably first appeared in connection with the λ-calculus. It was called a reduction relation, and the clauses of its definition were called reduction rules. Gordon Plotkin first made me aware of the power and flexibility of such relations in giving meaning-by-evaluation to programming languages. (In passing we may note that the original definition of ALGOL 68, though strongly verbal, is in essence a set of reduction rules.)

Inaction

NIL has no atomic actions.

Summation

From $B_1 \xrightarrow{\mu v} B_1'$ infer $B_1 + B_2 \xrightarrow{\mu v} B_1'$

From $B_2 \xrightarrow{\mu v} B_2'$ infer $B_1 + B_2 \xrightarrow{\mu v} B_2'$

Thus the atomic actions of a sum are exactly those of its summands. We adopt the following presentation of such inference rules:

Sum $\rightarrow$	(1) $\dfrac{B_1 \xrightarrow{\mu v} B_1'}{B_1 + B_2 \xrightarrow{\mu v} B_1'}$	(2) $\dfrac{B_2 \xrightarrow{\mu v} B_2'}{B_1 + B_2 \xrightarrow{\mu v} B_2'}$

Action

Act →

(1) $\alpha x_1,\ldots,x_n.B \xrightarrow{\alpha(v_1,\ldots,v_n)} B\{v_1/x_1,\ldots,v_n/x_n\}$

(2) $\bar{\alpha}v_1,\ldots,v_n.B \xrightarrow{\bar{\alpha}(v_1,\ldots,v_n)} B$

(3) $\tau.B \xrightarrow{\tau} B$

Notes: (i) These are not inference rules, but axioms.

(ii) Act → (1) holds for all tuples $(v_1,\ldots,v_n)$ (of appropriate type for α), while Act → (2) holds just for the tuple qualified by $\bar{\alpha}$.

(iii) See §5.5 below for why we consider only values $v_1,\ldots,v_n$ (not expressions $E_1,\ldots,E_n$) in Act → (2).

Composition

Com →

(1) $\dfrac{B_1 \xrightarrow{\mu v} B_1'}{B_1|B_2 \xrightarrow{\mu v} B_1'|B_2}$ (2) $\dfrac{B_2 \xrightarrow{\mu v} B_2'}{B_1|B_2 \xrightarrow{\mu v} B_1|B_2'}$

(3) $\dfrac{B_1 \xrightarrow{\lambda v} B_1' \qquad B_2 \xrightarrow{\bar{\lambda} v} B_2'}{B_1|B_2 \xrightarrow{\tau} B_1'|B_2'}$

Notes: (i) Com → (1) and (2) express the idea that an action of B_1 or of B_2 in the composition $B_1|B_2$ yields an action of the composite in which the other component is unaffected.

(ii) Com → (3) expresses that communication of components yields a τ-action of the composite.

Restriction

Res →	$\dfrac{B \xrightarrow{\mu V} B'}{B\backslash\alpha \xrightarrow{\mu V} B'\backslash\alpha}$, $\mu \notin \{\alpha, \bar{\alpha}\}$

Note: the side condition ensures that $B\backslash\alpha$ has no αv or $\bar{\alpha} v$ actions.

Relabelling

Rel →	$\dfrac{B \xrightarrow{\mu V} B'}{B[S] \xrightarrow{(S\mu)V} B'[S]}$

Note: recall our convention that $S\tau = \tau$.

Identifier. Suppose that identifier b is defined by the (possibly recursive) clause

$$b(x_1,\ldots,x_{n(b)}) \Longleftarrow B_b \qquad (FV(B_b) \subseteq \{x_1,\ldots,x_{n(b)}\})$$

We shall discuss such definitions shortly. Our rule is

Ide →	$\dfrac{B_b\{v_1/x_1,\ldots,v_{n(b)}/x_{n(b)}\} \xrightarrow{\mu V} B'}{b(v_1,\ldots,v_{n(b)}) \xrightarrow{\mu V} B'}$

Note: the rule says, in effect, that each parameterized identifier has exactly the same actions as the appropriate instance of the right-hand side of its definition.

Conditional

Con →	(1) $\dfrac{B_1 \xrightarrow{\mu V} B_1'}{\underline{\text{if}}\ \text{true}\ \underline{\text{then}}\ B_1\ \underline{\text{else}}\ B_2 \xrightarrow{\mu V} B_1'}$	(2) $\dfrac{B_2 \xrightarrow{\mu V} B_2'}{\underline{\text{if}}\ \text{false}\ \underline{\text{then}}\ B_1\ \underline{\text{else}}\ B_2 \xrightarrow{\mu V} B_2'}$	

Note: As with all value expressions without variables, we assume that boolean-valued expressions evaluate 'automatically' to their boolean values. See §5.5 below for why we need not consider value-expressions containing variables in these rules.

5.4 Defining behaviour identifiers

We shall now assume that every behaviour identifier b is defined by a clause

$$b(x_1,\ldots,x_{n(b)}) \Leftarrow B_b$$

where $x_1,\ldots,x_{n(b)}$ are distinct variables, and where $FV(B_b) \subseteq \{x_1,\ldots,x_{n(b)}\}$. The symbol '$\Leftarrow$' is preferred to ' = ' since we are not yet talking of the behaviours denoted by behaviour expressions (so ' = ', in the sense of equality of meaning, would be out of place), and also because we will later in this chapter use ' =' to mean identity between expressions.

We thus have a collection of clauses defining our b's, and they may be mutually recursive. Although not actually essential, we shall impose a slight constraint on the collection, which will forbid such definitions as

$$b(x) \Leftarrow \bar{\alpha}x.NIL + b(x+1)$$

or

$$\begin{cases} b_1 \Leftarrow b_2 + \alpha.b_3 \\ b_2 \Leftarrow b_1 \mid \beta.b_4 \end{cases}$$

in which a behaviour may 'call itself recursively without passing a guard'. Thus the following are permitted:

$$b(x) \Leftarrow \bar{\alpha}x.NIL + \tau.b(x+1)$$

and

$$\begin{cases} b_1 \Leftarrow b_2 + \alpha.b_3 \\ b_2 \Leftarrow \tau.b_1 \mid \beta.b_4 \end{cases}$$

More precisely, we say that b is unguarded in B if it occurs in B without an enclosing guard. The restriction on our defining clauses for the b's is that there must be no infinite sequence $b_{i(1)},b_{i(2)},\ldots$ such that, for each j, $b_{i(j+1)}$ is unguarded in $b_{i(j)}$. (In the forbidden examples above there are such sequences: $b,b,b,\ldots$ and $b_1,b_2,b_1,b_2,\ldots$ respectively.) Further, for correctness of sorts, we require

$$L(B_b) \subseteq L(b)$$

When the above constraints are met, we shall say that the behaviour identifiers are guardedly well-defined.

5.5 Sorts and programs

Our formation rules ascribe a unique sort L(B) each behaviour expression B ; we write

$$B : L(B)$$

to mean 'B possesses sort L(B)' . For many reasons, it is convenient to allow B to possess all larger sorts as well; so we declare

$$B : L \;\&\; L \subseteq M \text{ implies } B : M$$

For example, this allows us to make sense of an expression like

$$NIL[\beta/\alpha]$$

since $\beta/\alpha : \{\alpha\} \to \{\beta\}$ is a relabelling, and $NIL : \{\alpha\}$ since $NIL : \emptyset$.

An important property of atomic actions as defined in §5.3 is the following:

Proposition 5.1 If $B \xrightarrow{\mu V} B'$, and $B : L$, then

$$\mu \in L \cup \{\tau\} \text{ and } B' : L$$

Proof By induction on the length of the inference which ensures $B \xrightarrow{\mu V} B'$, using the ascription of sorts by the formation rules. □

Although our rules for atomic actions apply to arbitrary behaviour expressions, they fail to describe fully the meaning of expressions with free variables. For example, the rule Act→ gives no action for

$$\bar{\alpha}(x+1).\ NIL$$

and Con→ says nothing for

$$\underline{\text{if}}\ x \geq 0\ \underline{\text{then}}\ \bar{\alpha}x.NIL\ \underline{\text{else}}\ \bar{\beta}(-x).NIL$$

Clearly they could not determine the actions of these expressions, since actions involve values, not variables, and in the second example even the label of the possible action depends upon the 'value' of x .

We choose to regard the meaning of a behaviour expression B with free variables $\tilde{x}$ as determined by the meanings of $B\{\tilde{v}/\tilde{x}\}$ for all value-vectors $\tilde{v}$.

Definition We define a program to be a closed behaviour expression, i.e. one with no free variables.

Now the fact that our rules describe the meanings of programs satisfactorily is due to the following:

Proposition 5.2 If B is a program and $B \xrightarrow{\mu v} B'$, then B' is also a program.

Proof By induction on the length of the inference which ensures $B \xrightarrow{\mu v} B'$. The condition on the free variables of each B_b, and the substitution involved in Act→(1), are critical. ☒

5.6 Direct equivalence of behaviour programs

(In §5.6 and §5.7 we are concerned only with programs).

We now take up the question, posed in §5.1, of which behaviour programs possess the same derivations; this will yield an equivalence relation, which will also be a congruence - that is, any program may be replaced by an equivalent one in any context, without affecting the behaviour (derivations) of the whole. For example,

$$B+B' \qquad \text{and} \qquad B'+B$$

are different programs, but we clearly expect them to be interchangeable in this sense.

A first approximation to what we want may be called direct equivalence; we denote it by $\equiv$, and define it as follows:

Definition $B_1 \equiv B_2$ (B_1 and B_2 are directly equivalent) iff for every μ, v and B

$$B_1 \xrightarrow{\mu v} B \Leftrightarrow B_2 \xrightarrow{\mu v} B .$$

(Warning: $\equiv$ is not a congruence relation. For example, we may have $B_1 \equiv B_2$, but in general

$B \mid B_1 \not\equiv B \mid B_2$. For example,

$$\left.\begin{array}{l} \alpha.NIL \mid B_1 \xrightarrow{\alpha} NIL \mid B_1 \\ \alpha.NIL \mid B_2 \xrightarrow{\alpha} NIL \mid B_2 \end{array}\right\} \text{ not identical!}$$

But the congruence relation we want will be implied by $\equiv$, and so the following laws for $\equiv$ will hold for the congruence also.)

In what follows it is often convenient to let g stand for an arbitrary guard $\alpha\tilde{x}$, $\bar{\alpha}\tilde{E}$ or τ. The result Sg of relabelling a guard is given by $S(\alpha\tilde{x}) = (S\alpha)\tilde{x}$, $S(\bar{\alpha}\tilde{E}) = (S\bar{\alpha})\tilde{E}$ and $S\tau = \tau$. The name of the label in g is denoted by $name(g)$.

Theorem 5.3 (Direct Equivalences). The following direct equivalences hold (classified by the leading operator on the left side):

Sum $\equiv$ (1) $B_1 + B_2 \equiv B_2 + B_1$ (2) $B_1 + (B_2 + B_3) \equiv (B_1 + B_2) + B_3$
(3) $B + NIL \equiv B$ (4) $B + B \equiv B$

Act $\equiv$ $\alpha\tilde{x}.B \equiv \alpha\tilde{y}.B\{\tilde{y}/\tilde{x}\}$ (change of bound variables)
where $\tilde{y}$ are distinct variables not in B

Res $\equiv$ (1) $NIL\backslash\beta \equiv NIL$ (2) $(B_1 + B_2)\backslash\beta \equiv B_1\backslash\beta + B_2\backslash\beta$
(3) $(g.B)\backslash\beta \equiv \begin{cases} NIL & \text{if } \beta = name(g) \\ g.B\backslash\beta & \text{otherwise} \end{cases}$

Rel $\equiv$ (1) $NIL[S] \equiv NIL$ (2) $(B_1 + B_2)[S] \equiv B_1[S] + B_2[S]$
(3) $(g.B)[S] \equiv Sg.B[S]$

Now in view of **Sum** $\equiv$ the following notations are unambiguous:

$\sum_{1 \le i \le n} B_i$ meaning $B_1 + \ldots + B_n$ (NIL, if $n = 0$)

$\sum\{B_i ; i \in I\}$ more generally, where I is finite.

If each B_i is of form $g_i.B_i'$, we call such a sum a sum of guards, and each B_i a summand.

Com $\equiv$ Let B and C be sums of guards. Then

$$\begin{aligned} B|C \equiv\ & \sum\{g.(B'|C) ;\ g.B' \text{ a summand of } B\} \\ & + \sum\{g.(B|C') ;\ g.C' \text{ a summand of } C\} \\ & + \sum\{\tau.(B'\{\tilde{v}/\tilde{x}\}|C') ;\ \alpha\tilde{x}.B' \text{ a summand of } B \\ & \qquad \text{and } \bar{\alpha}\tilde{v}.C' \text{ a summand of } C\} \\ & + \sum\{\tau.(B'|C'\{\tilde{v}/\tilde{x}\}) ;\ \bar{\alpha}\tilde{v}.B' \text{ a summand of } B \\ & \qquad \text{and } \alpha\tilde{x}.C' \text{ a summand of } C\} \end{aligned}$$

Ide ≡	Let identifier b be defined by $b(\tilde{x}) \Leftarrow B_b$; then $b(\tilde{v}) \equiv B_b\{\tilde{v}/\tilde{x}\}$

Con ≡	(1) if true then B_1 else $B_2 \equiv B_1$ (2) if false then B_1 else $B_2 \equiv B_2$

Proof To prove each law is a routine application of the definition of the relations $\xrightarrow{\mu V}$. We consider three laws:

(i) Sum ≡ (2): $B_1+(B_2+B_3) \equiv (B_1+B_2)+B_3$

Let $B_1+(B_2+B_3) \xrightarrow{\mu V} B$. This can only be due to

either rule Sum → (1), because $B_1 \xrightarrow{\mu V} B$

or rule Sum → (2), because $B_2+B_3 \xrightarrow{\mu V} B$,

and in the latter case, similarly, either $B_2 \xrightarrow{\mu V} B$ or $B_3 \xrightarrow{\mu V} B$.

In each of the three cases, rules Sum → (1) and Sum → (2) yield

$$(B_1+B_2)+B_3 \xrightarrow{\mu V} B .$$

The reverse implication is similar.

(ii) Res ≡ (3) : $(\alpha\tilde{x}.B)\backslash\beta \equiv \begin{cases} \text{NIL} & (\beta=\alpha) \\ \alpha\tilde{x}.(B\backslash\beta) & (\beta\neq\alpha) \end{cases}$

By Act → (1) , the only actions of $\alpha\tilde{x}.B$ are of form

$$\alpha\tilde{x}.B \xrightarrow{\alpha\tilde{v}} B\{\tilde{v}/\tilde{x}\} \quad \text{(for arbitrary } \tilde{v}\text{)}.$$

Thus $(\alpha\tilde{x}.B)\backslash\alpha$ has no actions (since Res → yields none) ;

neither has NIL, which settles the case $\beta = \alpha$.

For $(\beta \neq \alpha)$, by Res → the only actions of $(\alpha\tilde{x}.B)\backslash\beta$ are

$$(\alpha\tilde{x}.B)\backslash\beta \xrightarrow{\alpha\tilde{v}} B\{\tilde{v}/\tilde{x}\}\backslash\beta = (B\backslash\beta)\{\tilde{v}/\tilde{x}\}$$

and these are exactly the actions of $\alpha\tilde{x}.(B\backslash\beta)$.

The proof for guards $\overline{\alpha}\tilde{v}$ and τ is similar.

(iii) Com ≡ : $B|C \equiv \sum\cdots+\sum\cdots+\sum\cdots+\sum\cdots$.

(We use X to abbreviate the right-hand side.)

Let $B|C \xrightarrow{\mu V} D$. There are several cases.

(a) $B \xrightarrow{\mu V} B''$, and $D = B''|C$ (by Com → (1)).

Then B has a summand $g.B'$ for which $g.B' \xrightarrow{\mu V} B''$

(by Sum →). This action must be an instance of Act →

from which we can also find that $g.(B'|C) \xrightarrow{\mu v} B''|C$
(considering the three types of guard).
Hence also $X \xrightarrow{\mu v} B''|C = D$.

(b) $C \xrightarrow{\mu v} C''$, and $D = B|C''$ (by Com→(2))
The argument that $X \xrightarrow{\mu v} D$ is similar.

(c) $B \xrightarrow{\alpha \tilde{u}} B''$, $C \xrightarrow{\bar{\alpha} \tilde{u}} C'$ and $\mu v = \tau$, $D = B''|C'$
(by Com→(3) ; there is a similar case with $\alpha, \bar{\alpha}$ exchanged)
Then by Sum→ and Act→ , B has a summand $\alpha \tilde{x}.B'$
and $B'' = B'\{\tilde{u}/\tilde{x}\}$, while C has a summand $\bar{\alpha} \tilde{u}.C'$.
Hence, since X has a summand $\tau.(B'\{\tilde{u}/\tilde{x}\}|C')$, we have

$$X \xrightarrow{\tau} B''|C' = D \text{ , as required .}$$

We have now shown by (a), (b) & (c) that for all μ, v and D

$$B|C \xrightarrow{\mu v} D \Longrightarrow X \xrightarrow{\mu v} D$$

and the reverse implication can be argued similarly.

□

Exercise 5.1 Prove some more of the equivalences claimed;
e.g. Sum ≡(1), Res ≡(2), Rel ≡(2) and Con ≡(1) . They are all as easy as Sum ≡(2) .

5.7 Congruence of behaviour programs

We now propose to extend or widen our direct equivalence relation to a congruence relation. Apart from the wish to get a congruence relation (so that equivalence is preserved by substitution of equivalent programs) there is another motivation; ' ≡ ' requires that the results of actions of equivalent programs should be identical, and it is reasonable to ask only that the results should be equivalent again.

We therefore define the relation ' ~ ' over programs, which we call strong equivalence (we define it analogously to the observation equivalence of §3.3, but it is stronger because we do not allow arbitrary τ-actions to interleave the observable actions). We define it in terms of a decreasing sequence $\sim_0, \sim_1, \ldots, \sim_k, \ldots$ of equivalence relations:

Definition $B \sim_0 C$ is always true;

$B \sim_{k+1} C$ iff for all μ, v

(i) if $B \xrightarrow{\mu v} B'$ then for some C', $C \xrightarrow{\mu v} C'$ and $B' \sim_k C'$,

(ii) if $C \xrightarrow{\mu v} C'$ then for some B', $B \xrightarrow{\mu v} B'$ and $B' \sim_k C'$;

$B \sim C$ iff $\forall k \geq 0.\ B \sim_k C$ (i.e. $\sim = \bigcap_k \sim_k$).

We leave out the simple proofs that each $\sim_k$ is an equivalence relation, and that $B \sim_{k+1} C$ implies $B \sim_k C$ (i.e. the sequence of equivalences is decreasing).

Exercise 5.2 Show that $B \equiv C$ implies $B \sim_k C$ for each k, and hence implies $B \sim C$.

Theorem 5.4 $\sim$ is a congruence relation.

More precisely, $B_1 \sim B_2$ implies

$$B_1 + C \sim B_2 + C, \quad C + B_1 \sim C + B_2$$
$$\tilde{\alpha}\,\tilde{v}.B_1 \sim \tilde{\alpha}\tilde{v}.B_2, \quad \tau.B_1 \sim \tau.B_2$$
$$B_1 | C \sim B_2 | C, \quad C | B_1 \sim C | B_2$$
$$B_1 \backslash \alpha \sim B_2 \backslash \alpha, \quad B_1[S] \sim B_2[S]$$

and $B_1\{\tilde{v}/\tilde{x}\} \sim B_2\{\tilde{v}/\tilde{x}\}$ (for all $\tilde{v}$) implies

$$\alpha\tilde{x}.B_1 \sim \alpha\tilde{x}.B_2$$

Proof We give the proof only for composition. We prove by induction on k that

$$B_1 \sim_k B_2 \text{ implies } B_1 | C \sim_k B_2 | C$$

For $k = 0$ it is trivial. Now assume $B_1 \sim_{k+1} B_2$.

Let $B_1 | C \xrightarrow{\mu v} D_1$. We want D_2 such that

$$B_2 | C \xrightarrow{\mu v} D_2 \sim_k D_1$$

There are three cases:

(a) $B_1 \xrightarrow{\mu v} B_1'$, and $D_1 = B_1' | C$ (by Com→(1))

Then $B_2 \xrightarrow{\mu v} B_2' \sim_k B_1'$ for some B_2',

whence $B_2 | C \xrightarrow{\mu v} B_2' | C$ by Com→(1)

$\sim_k D_1 (= B_1' | C)$ by inductive hypothesis.

(b) $C \xrightarrow{\mu v} C'$ and $D_1 = B_1 | C'$ (by Com→(2))

Then $B_2 | C \xrightarrow{\mu v} B_2 | C'$ by Com→(2)

But $B_1 \sim_k B_2$ (since $B_1 \sim_{k+1} B_2$), hence $B_1 | C' \sim_k B_2 | C'$ by inductive hypothesis.

(c) $B_1 \xrightarrow{\lambda\tilde{u}} B_1'$, $C \xrightarrow{\bar{\lambda}\tilde{u}} C'$ and $\mu v = \tau$, $D_1 = B_1'|C'$ (by Com→(3))
Then $B_2 \xrightarrow{\lambda\tilde{u}} B_2' \sim_k B_1'$, for some B_2' ,
whence $B_2|C \xrightarrow{\tau} B_2'|C'$ by Com→(3)

$\sim_k D_1$ by inductive hypothesis .

By symmetry, of course, if $B_2|C \xrightarrow{\mu v} D_2$ then we find D_1 such that $B_1|C \xrightarrow{\mu v} D_1 \sim_k D_2$. ☒

Exercise 5.3 (i) Prove that $B_1 \sim_k B_2$ implies $\tilde{\alpha}\tilde{v}.B_1 \sim_{k+1} \tilde{\alpha}\tilde{v}.B_2$; this shows that $B_1 \sim B_2$ implies $\tilde{\alpha}\tilde{v}.B_1 \sim \tilde{\alpha}\tilde{v}.B_2$, and also that guarding increases the index of $\sim_k$ by one.

(ii) Prove the last part of the Theorem, involving the positive label guard.

We end this section by giving some useful properties of $\sim$, in the form of equational laws. Note that Theorem 5.3 already gives many of its properties, since $\equiv$ is contained in $\sim$. Since we run the risk of bewildering the reader with a confused mass of properties, let us emphasize some structure.

In Theorem 5.3, Sum $\equiv$ states that + and NIL form a commutative semigroup with absorption, and Res $\equiv$, Rel $\equiv$, Com $\equiv$ each describe how one of the static behaviour operations $\backslash\alpha$, $[S]$, $|$ interacts with the dynamic operations +, μv and NIL. In the following theorem Com$\sim$ states that $|$ and NIL form a commutative semigroup, while Res$\sim$ and Rel$\sim$ state how the static operations interact with each other. The laws of Theorem 5.5 are only concerned with the static operations- they are essentially the Laws of Flow in [MM, Mil 2] .

Theorem 5.5 (Strong congruences) The following strong congruences hold:

Com$\sim$	(1) $B_1\|B_2 \sim B_2\|B_1$	(2) $B_1\|(B_2\|B_3) \sim (B_1\|B_2)\|B_3$
	(3) $B\|NIL \sim B$	

Res$\sim$	(1) $B\backslash\alpha \sim B$	$(B:L,\ \alpha \notin names(L))$
	(2) $B\backslash\alpha\backslash\beta \sim B\backslash\beta\backslash\alpha$	
	(3) $(B_1\|B_2)\backslash\alpha \sim B_1\backslash\alpha\|B_2\backslash\alpha$	$(B_1:L_1, B_2:L_2, \alpha \notin names(L_1 \cap \bar{L}_2))$

Rel$\sim$ (1) $B[I] \sim B$ ($I:L \to L$ is the identity relabelling)

(2) $B[S] \sim B[S']$ ($B:L$, and $S\restriction L = S'\restriction L$)

(3) $B[S][S'] \sim B[S' \circ S]$

(4) $B[S]\backslash\beta \sim B\backslash\alpha[S]$ ($\beta = \mathrm{name}(S(\alpha))$)

(5) $(B_1 | B_2)[S] \sim B_1[S] \,|\, B_2[S]$

Proof We give the proof of Com$\sim$(2). It is the hardest - but all the proofs are routine inductions.

We prove $\forall B_1 B_2 B_3 \,.\, B_1 | (B_2 | B_3) \sim_k (B_1 | B_2) | B_3$ by induction on k.

For $k = 0$ it's trivial.

Now for $k+1$, let $B_1 | (B_2 | B_3) \xrightarrow{\mu v} D$; we require D' such that $(B_1 | B_2) | B_3 \xrightarrow{\mu v} D' \sim_k D$.

There are several cases:

(a) $B_1 \xrightarrow{\mu v} B_1'$, and $D = B_1' | (B_2 | B_3)$ by Com$\to$(1).
Then $(B_1 | B_2) | B_3 \xrightarrow{\mu v} (B_1' | B_2) | B_3$ by Com$\to$(1) twice
$\sim_k D$ by induction.

(b) $B_2 | B_3 \xrightarrow{\mu v} C$, and $D = B_1 | C$ by Com$\to$(2).

Subcases

(i) $B_2 \xrightarrow{\mu v} B_2'$, and $C = B_2' | B_3$ by Com$\to$(1) ; i.e. $D = B_1 | (B_2' | B_3)$.
Then $B_1 | B_2 \xrightarrow{\mu v} B_1 | B_2'$ by Com$\to$(2),
so $(B_1 | B_2) | B_3 \xrightarrow{\mu v} (B_1 | B_2') | B_3$ by Com$\to$(1),
$\sim_k D$ by induction.

(ii) $B_3 \xrightarrow{\mu v} B_3'$, and $C = B_2 | B_3'$ by Com$\to$(2); similar.

(iii) $B_2 \xrightarrow{\lambda u} B_2'$, $B_3 \xrightarrow{\bar{\lambda} u} B_3'$, $C = B_2' | B_3'$ and $\mu v = \tau$;
so $D = B_1 | (B_2' | B_3')$ by Com$\to$(3).
Then $B_1 | B_2 \xrightarrow{\lambda u} B_1 | B_2'$ by Com$\to$(2),
so $(B_1 | B_2) | B_3 \xrightarrow{\tau} (B_1 | B_2') | B_3'$ by Com$\to$(3),
$\sim_k D$ by induction.

(c) $B_1 \xrightarrow{\lambda u} B_1'$, $B_2 | B_3 \xrightarrow{\bar{\lambda} u} C$, $D = B_1' | C$ and $\mu v = \tau$ by Com$\to$(3).

Subcases

(i) $B_2 \xrightarrow{\bar{\lambda} u} B_2'$, and $C = B_2' | B_3$ by Com$\to$(1) ; i.e. $D = B_1' | (B_2' | B_3)$
Then $B_1 | B_2 \xrightarrow{\tau} B_1' | B_2'$ by Com$\to$(3),
so $(B_1 | B_2) | B_3 \xrightarrow{\tau} (B_1' | B_2') | B_3$ by Com$\to$(1),
$\sim_k D$ by induction.

(ii) $B_3 \xrightarrow{\bar{\lambda} u} B_3'$, and $C = B_2 | B_3'$ by Com$\to$(2) : similar.

Thus we have found the required $D' \sim_k D$ in each case;
Similarly given $(B_1|B_2)|B_3 \xrightarrow{\mu v} D$, we find D' such that

$$B_1|(B_2|B_3) \xrightarrow{\mu v} D' \sim_k D .$$

This completes the inductive step, showing

$$B_1|(B_2|B_3) \sim_{k+1} (B_1|B_2)|B_3 \qquad \square$$

Exercise 5.4 Prove Com~(3) and Res~(3). For the second, you need to appeal to Proposition 5.1.

We now state and prove a theorem which we need later. It depends critically on the assumption that all behaviour identifiers are guardedly well defined (§5.4).

Theorem 5.6 Strong congruence 'satisfies its definition' in the following sense:

$B \sim C$ iff for all μ, v

(i) if $B \xrightarrow{\mu v} B'$ then for some C', $C \xrightarrow{\mu v} C'$ and $B' \sim C'$,

(ii) if $C \xrightarrow{\mu v} C'$ then for some B', $B \xrightarrow{\mu v} B'$ and $B' \sim C'$.

Proof ($\Leftarrow$) $B' \sim C'$ implies $B' \sim_k C'$ for any k; hence from (i) and (ii) we deduce $B \sim_{k+1} C$ for all k, by definition, whence $B \sim C$.

($\Rightarrow$) Since $B \sim_{k+1} C$ for all k, we have by definition that if $B \xrightarrow{\mu v} B'$ then, for each k, $\exists C_k . C \xrightarrow{\mu v} C_k$ & $B' \sim_k C_k$. But from our assumption that all behaviour identifiers are guardedly well-defined it follows that $\{C' ; C \xrightarrow{\mu v} C'\}$ is finite (we omit the details of this argument). Hence for some C',

$$C \xrightarrow{\mu v} C' \text{ and } B' \sim_k C' \text{ for infinitely many } k$$

and this implies $B' \sim_k C'$ for all k, since the relations $\sim_k$ are decreasing in k, hence $B' \sim C'$.

Thus (i) is proved, and (ii) is similar. $\square$

5.8 Congruence of Behaviour expressions and the Expansion Theorem

Having established definitions and properties of direct equivalence and congruence of programs - behaviour expressions without free variables - we are now in a position to lift the results to arbitrary behaviour expressions.

All that is needed is to define $\equiv$ and $\sim$ over expressions as follows:

Definition

Let $\tilde{x}$ be the free variables occurring in B_1 or B_2 or both. Then

$B_1 \equiv B_2$ iff, for all $\tilde{v}$, $B_1\{\tilde{v}/\tilde{x}\} \equiv B_2\{\tilde{v}/\tilde{x}\}$

$B_1 \sim B_2$ iff, for all $\tilde{v}$, $B_1\{\tilde{v}/\tilde{x}\} \sim B_2\{\tilde{v}/\tilde{x}\}$

Now we clearly want to extend the results of Theorems 5.3, 5.5 to arbitrary expressions; for example, we would like to apply Com~(3) of Theorem 5.5 to replace

$$\bar{\alpha}(x+1).\,\mathrm{NIL}\,|\,\mathrm{NIL} \quad \text{by} \quad \bar{\alpha}(x+1).\,\mathrm{NIL}$$

anywhere in any expression, but the law only applies at present to programs, and the expressions shown have a free variable x.

We state without proof the desired generalisation.

Theorem 5.7 The relation $\sim$ is a congruence over behaviour expressions. Moreover, the results of Theorems 5.3, 5.5 hold over arbitrary expressions, with the following adjustments:

(i) In Com≡ and Ide≡ of Theorem 5.3, replace $\tilde{v}$ (a value tuple) everywhere by $\tilde{E}$ (a tuple of value expressions).

(ii) Add in Com≡ the condition that, in the first (resp.second) sum on the right-hand side, no free variable of C(resp. B) is bound by g.

□

We now have enough to prove the Expansion Theorem, which we used in Chapter 4.

Theorem 5.8 (The Expansion Theorem).

Let $B = (B_1|\ldots|B_m)\backslash A$, where each B_i is a sum of guards. Then

$$B \sim \sum\{g.((B_1|\ldots|B_i'\ \ldots|B_m)\backslash A);\ g.B_i' \mid \text{a summand of } B_i,\ \mathrm{name}(g) \notin A\}$$
$$+ \sum\{\tau.((B_1|\ldots|B_i'\{\tilde{E}/\tilde{x}\}|\ldots|B_j'|\ldots|B_m)\backslash A);\ \alpha\tilde{x}.B_i' \text{ a summand of } B_i,\ \bar{\alpha}\tilde{E}.B_j' \text{ a summand of } B_j,\ i \neq j\}$$

provided that in the first term no free variable in $B_k (k \neq i)$ is bound by g.

Proof. We first show, by induction on m, that

$$B_1|\ldots|B_m \sim \sum\{g.(B_1|\ldots|B_i'|\ldots|B_m) \;;\; g.B_i' \text{ a summand of } B_i \;,\; 1 \le i \le m\}$$
$$+ \sum\{\tau.(B_1|\ldots|B_i'\{\tilde{E}/\tilde{x}\}|\ldots|B_j'|\ldots|B_m) \;;\; \alpha\tilde{x}.B_i' \text{ a summand of } B_i \;,\; \bar{\alpha}\tilde{E}.B_j' \text{ a summand of } B_j \;,\; i,j \in \{1,\ldots,m\},\; i \neq j\}$$

under the proviso of the Theorem. Note first that for $m = 1$ the second term is vacuous and the result follows simply by reflexivity of $\sim$. Now assume the property for $m-1$, with right-hand side C. Then we have (by congruence)

$$B_1|\ldots|B_{m-1}|B_m \sim C|B_m$$

and we may apply Com $\equiv$, generalised as in Theorem 5.7, since each of C and B_m is a sum of guards - and moreover the side-condition for Com $\equiv$ (stated as (ii) in Theorem 5.7) follows from the proviso of the present theorem. The property for m then follows by routine, though slightly tedious, manipulations; of course we rely strongly on Com $\sim$ (2).

Finally, the theorem follows easily by repeated use of Res $\equiv$(3) and Sum $\equiv$(3). ∎

Exercise 5.5 Complete the details of the inductive step in the proof, and see exactly where the proviso of the theorem is necessary.

In summary : we now have a powerful set of laws for transforming programs and behaviour expressions while preserving their derivation pattern. (These laws are enough to prove the Expansion Theorem, Theorem 5.8, for example.)

We have prepared the way for introducing CTs, an algebra which satisfies these laws and so may be regarded as a model of CCS which is faithful to its derivation patterns.

But we should mention that observation equivalence ($\approx$) (generalised from §3.3 to admit value-passing) is a wider relation that our $\sim$, and satisfies still more equational laws.