THE FORMAL SPECIFICATION

OF A CONFERENCE ORGANIZING SYSTEM

T. Clement

Technical Monograph PRG-36
August 1983

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD

Address from September 1983:

Syracuse University
School of Computer and Information Science
313 Link Hall
Syracuse
New York 13210
U S A.

1.1 The problem

This monograph arises from an exercise in information systems design sponsored by IFIP Working Group 8.1 ([Olle82]). This was intended to provide a means of comparing various design methods, proprietary and otherwise, by asking their exponents to prepare designs from a common specification to a level where the system could be produced by a typical applications programmer.

The problem as posed by IFIP is to produce an information system to support the organization of a conference. They explain that working conferences are normally run by a programme committee, responsible for the technical content, and an organizing committee, who are responsible for finance, publicity, and the hotel and conference facilities required by delegates. This clearly involves cooperation between the committees, and a substantial amount of shared information to be kept up to date.

The description of the existing organization identifies various activities which the committees perform, and with which assistance is sought. The programme committee invites contributions, and receives declarations of intention to submit papers. It also registers the receipt of papers, arranges their refereeing, collates the reports and selects papers, and finally draws up a timetable, with appropriate people chosen to chair each session.

Meanwhile, the organizing committee is responsible for issuing invitations to attend the conference, registering replies, and ensuring that the conference does not grow too large for its accomodation. It is their policy to invite authors of papers, as well as certain categories of IFIP official. They occasionally send several invitations to the same person if they come into more than one of these categories, and they would like the system

to spare them this embarrassment They also prepare a final list of attendees Other aspects of the work of the committees are not detailed in the exercise, and were not required as part of the submitted design.

1 2 The goals of formal specification

As producers of computing systems, we find ourselves with the insoluble problem of creating objects whose behaviour, which can be objectively tested, is to correspond to some partial description of a system which we do not fully understand. Our approach is to divide it into two: to find means of implementing precisely a precise specification, and means of agreeing a specification in precise form with our client before constructing a complete system.

Our attack on the first of these problems is through the application of formal notation a mathematical system in which every statement and all the means of manipulation are precisely defined, so that unambiguous specifications can be written, and proposed implementations can be checked for consistency with them, at least in principle.

The second problem remains insoluble, and is one common to all the professions: the lawyer knows the law, but not the client; the doctor knows the body but not the patient; we know about constructing computer systems, but not about the particular application. At the same time, the clients are seeking professional help only because they do not have the knowledge (of the law, of medicine, of computing) to help themselves. They do not understand what information they should give to help their case

The doctor resolves the situation by posing questions of the patient which are inspired by an internal model of the body The purpose of a systems design methodology for the designer is to provide a pattern for systems in general within which the nature of a particular system can be determined

Such a pattern need not even be explicit. If it is, it may be expressed formally or informally. The more detailed it is, the more specific the questions may be, but the fewer the systems it will describe. We should therefore be prepared to deploy a variety of patterns, in the light of our knowledge of their areas of applicability. So far, we have developed them as the problems have arisen, so each is a prototype to be prepared and tested in the course of a specification.

The completed specification serves as a summary of the discussions which led to its creation. The clients should be confident that it represents what they want, and the designer that a system with the given properties exists, even under extra constraints such as a particular choice of machine, budget limitations, speed of response and so on. It can therefore be entered as part of the contract for the supply of the system.

1.3 The notation and the theory

Our starting point for the formal content of this specification is set theory: specifically the axiomatic set theory of [Abrial81]. Its attractions for our purposes are the ease with which the terms can be given informal interpretations (naive set theory) and its expressive power, which is adequate: consistency of much of mathematics has been shown by exhibiting set theoretic models.

The whole of Abrial's set theory is expressed as a syntax for a language in which its terms and predicates are written (a few lines of BNF or similar notation), and eighteen axioms and a proof rule which together allow us to categorize some of the sentences of the language as theorems. The specification of an informally described system is the inverse process to that of informally interpreting a formal system (for example, giving the naive intuitive explanation of a term in axiomatic set theory). We write sentences which incorporate the words used in the informal description as identifiers.

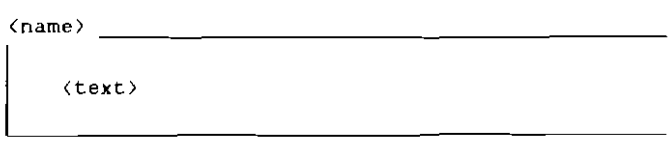
and which express properties that these identifiers possess. So far as the theory is concerned, these properties are not (necessarily) deducible, but axiomatic the result is a new theory, containing the new identifiers as constants, in which new theorems can be deduced It must be checked at two levels. The first check is for internal consistency: it is possible to introduce mutually inconsistent axioms and the result is a theory where everything is a theorem. The second is to relate the theory to the informal specification through the interpretation of identifiers. If the interpretation of a theorem corresponds to an untruth in the interpretation, then the theory is not consistent with reality, conversely, if some property of the real world cannot be shown as a theorem then the theory is incomplete, and admits models which do not have the property. (A simple repair is to add the property as an axiom!)

This completely describes the *principle* of our approach to specification. The *practical* problem in carrying out this programme is that the formally stated properties must be amenable to expression and manipulation, and so the strings of symbols which express them must be short, and the proofs which demonstrate their properties must also be short. Within the framework of the language, the method of attack is through abstraction. the nesting of the final theory inside other theories of greater generality, which introduce new constants to be used in constructing sentences, and axioms concerning these constants which can be used in reasoning about them. Some of this reasoning will be done immediately, to establish theorems of general utility. This method is evident in [Abrial81], in the successive introduction of functions finite sets, natural numbers, and sequences, each building on the previous ones

We also make use of some simple notations for generating mathematical text We define an identifier which expands into an arbitrary piece of mathematics (in many cases, of a form which could be associated with a quantifier) by

⟨name⟩ ≡ ⟨text⟩

or



The text may be modified by systematic substitution of identifiers by general terms, so that given, for example

$$XRANGE \triangleq t \in Y \rightarrow X$$

the predicate $XRANGE[Z \times Z/X]$ becomes $f \in Y \rightarrow Z \times Z$.

We shall save some effort by relaxing our rigour slightly. For example, we shall tend to move freely between the possible isomorphic bracketings of extended cross products. Informal notational conventions can also be convenient in particular, where auxiliary constants are introduced in definitions and theorems, we frequently find predicates of the form ⟨identifier⟩ $\in$ ⟨set⟩. Where the set is denoted by a single capital letter, for instance S, we shall assume that identifiers derived from the lower case letter, such as s, s', $\overline{s}$, s_1 and so on are members of it, unless otherwise indicated

2 A theoretical background

In the introduction, we identified two desires for a pattern for information systems to structure our dialogue with our client, and for a theory of information systems to lighten our notation for the specification. We bring the two together here: our pattern (the running informal text) will be the interpretation of the theory (the indented formal text). We shall proceed in three steps, from a consideration of the nature of information systems, through a notation for the commands that act upon them, to an approach to designing a practical system

2.1 Extending the theory for defining systems behaviour

The activities of many organizations depend on knowledge about the state of the world for example, a manufacturing company may decide to order more of a particular item when the number in the warehouse drops below some level The need for an information system (computer-based or not) arises when the direct observation of the state of the world is too difficult - for example, when the order clerk is too far from the warehouse to go and look on the appropriate shelf. The system being merely a simulacrum of part of the world, the obvious place to begin is in the specification of the part of the world which interests us, and its behaviour. This will determine the boundaries of the simulation, and we must be sure to make them explicit and be prepared to justify them: it seems unnecessary to record the colours of the warehouse shelves until we want to know if we have enough paint to repaint them.

The required observations are only one part of the specification. The state of the world is not static, but changing In many cases, this is not a continuous change, but a discrete one the stock level changes because an order is despatched (On the other hand, the volume of brandy maturing

in a cask decreases with time through evaporation) In the discrete case, we can maintain the simulation of the state by simulating the event in the information system There will normally be some time difference between the event and its simulation Its magnitude will not be specified formally here, but, given that the simulation reflects only the state resulting from the events in the system, we shall record the pious hope that it shall be 'as small as reasonably possible' Making the choice of events also sets boundaries to the system which we must justify. It may be reasonable to ignore the possibility of a warehouse fire, despite its effect on stock levels, but less reasonable to ignore deliveries.

Given the sets of events and observations, the specification must show how the observations are affected by the events, and the order in which events can occur. This ordering has an *implicit* effect on the possible observations for example, a person has a single social security number because the 'issue a number' event happens only once for each person Describing the effects of events may lead to the identifying of further events and observations to be included, and the final form that will be presented here is the result of many iterations of identifying events and specifying their effects. Doing this iteration at the specification stage is cheaper than waiting until implementation or test.

We need notations for presenting the temporal relationships between events and their effect on observations. We begin with the first, which we regard as establishing the patterns of behaviour in the world, and we shall restrict ourselves to the possible orderings in time There are cases where this is not adequate for example to record the constraint that goods paid for by cheque are not despatched for ten days, but this loss of expressive power is compensated for by a corresponding simplification of the theory.

The simplest behaviour is to do nothing before stopping. We give this special behaviour a name

SKIP ϵ BEHAVIOUR

Another fundamental behaviour of mainly technical interest does not even stop:

$$\mathrm{NULL} \in \mathrm{BEHAVIOUR}$$

The next most simple behaviour is to do one event once. We can identify the set of events, which we shall call E, with these behaviours.

$$E \subset \mathrm{BEHAVIOUR}$$

From these beginnings, we may then construct more complex behaviours. Given two behaviours, we may act like one *then* the other, one *or* the other, or one *and* the other in parallel. We introduce the three corresponding operators

$$\begin{aligned} ; &\in \mathrm{BEHAVIOUR} \times \mathrm{BEHAVIOUR} \rightarrow \mathrm{BEHAVIOUR} \\ [] &\in \mathrm{BEHAVIOUR} \times \mathrm{BEHAVIOUR} \rightarrow \mathrm{BEHAVIOUR} \\ ||\ &\in \mathrm{BEHAVIOUR} \times \mathrm{BEHAVIOUR} \rightarrow \mathrm{BEHAVIOUR} \end{aligned}$$

Conversely, if we take a particular behaviour, then we expect it to have been created from the raw materials of E and SKIP or NULL by use of these operators, rather than existing *a priori*. In an equational specification, these would be the generators of the sort, and in Larch ([Guttag83]) special syntax is used for introducing a rule of compulational induction over them. In our case, we can introduce this within the language

$$\begin{aligned} \mathrm{BEHAVIOUR} \Leftarrow\ & \{\mathrm{SKIP},\ \mathrm{NULL}\} \cup E \cup \\ & (\ ; \cup [] \cup ||\ )(\mathrm{BEHAVIOUR} \times \mathrm{BEHAVIOUR}) \end{aligned}$$

We can then define functions yielding the events which comprise any behaviour and the orders in which they can occur by recursion on the structure of that behaviour.

We may define the alphabet of a behaviour as the set of events which it involves, or formally as

$$\alpha \in \text{BEHAVIOUR} \rightarrow \mathbb{P}(E)$$

$$\alpha\,\text{SKIP} = \alpha\,\text{NULL} = \Phi_E \qquad \text{(A1)}$$
$$e \in E \Rightarrow \alpha\, e = \{e\} \qquad \text{(A2)}$$
$$\alpha(b_1;\ b_2) = \alpha(b_1 \,[]\, b_2) = \alpha(b_1 \,||\, b_2) = \alpha\, b_1 \cup \alpha\, b_2 \qquad \text{(A3)}$$

It will be helpful to formally define the pairs of behaviours which have no events in common

$$\text{Independent} \in \text{BEHAVIOUR} \leftrightarrow \text{BEHAVIOUR}$$
$$\text{Independent}(b_1,\ b_2) \Leftrightarrow \text{Disjoint}(\alpha\, b_1,\ \alpha\, b_2)$$

We can express the orderings of events as the possible sequences in which they can occur. Those sequences after which no more events may occur are defined by

$$\text{finals} \in \text{BEHAVIOUR} \rightarrow \mathbb{P}(\text{seq}(E))$$

$$\text{finals SKIP} = \{\langle\rangle\}$$
$$\text{finals NULL} = \Phi_{\text{seq}(E)}$$
$$\text{finals}\, e = \{\langle e\rangle\}$$

$$\text{finals}(b_1;\ b_2) = \{s^{\frown}t \mid s: \text{finals}\, b_1;\ t: \text{finals}\, b_2\}$$
$$\text{finals}(b_1 \,[]\, b_2) = \text{finals}\, b_1 \cup \text{finals}\, b_2$$
$$\text{finals}(b_1 \,||\, b_2) = \{t: \text{seq}(E) \mid \alpha\, t \subset \alpha(b_1 \,||\, b_2)\ \&$$
$$t \downarrow \alpha\, b_1 \in \text{finals}\, b_1\ \&$$
$$t \downarrow \alpha\, b_2 \in \text{finals}\, b_2\ \}$$

$$\text{traces} \in \text{BEHAVIOUR} \rightarrow \mathbb{P}(\text{seq}(E))$$

$$\text{traces SKIP} = \text{traces NULL} = \{\langle\rangle\}$$
$$\text{traces } e = \{\langle\rangle, \langle e\rangle\}$$

$$\text{traces}(b_1 ; b_2) =$$
$$\quad \text{traces } b_1 \cup \{s^\frown t \mid s : \text{finals } b_1 ; t : \text{traces } b_2\}$$
$$\text{traces}(b_1 \,[]\, b_2) = \text{traces}(b_1) \cup \text{traces}(b_2)$$
$$\text{traces}(b_1 \,||\, b_2) = \{t : \text{seq}(E) \mid \alpha t \subset \alpha(b_1 \,||\, b_2) \;\&$$
$$t \downarrow \alpha b_1 \in \text{traces } b_1 \;\&$$
$$t \downarrow \alpha b_2 \in \text{traces } b_2 \;\}$$

and from this we can derive the relationship between sequences of events and the events which can follow them in a behaviour, and the relationship between behaviours and the events which can start them.

$$\underline{\text{accepts}} \in \text{BEHAVIOUR} \rightarrow (\text{seq}(E) \leftrightarrow E)$$
$$t \; (\underline{\text{accepts}} \; b) \; e \Leftrightarrow t^\frown\langle e\rangle \in \text{traces } b$$

$$\underline{\text{starts}} \in E \leftrightarrow \text{BEHAVIOUR}$$
$$e \; \underline{\text{starts}} \; b \Leftrightarrow \langle\rangle \; (\underline{\text{accepts}} \; b) \; e$$

Are these definitions correct? They are clearly internally consistent, since we can find models for BEHAVIOUR, but do they have the properties we expect when we talk of these types of combination of behaviours? They have been carefully constructed to be reasonable, but we should like to have more confidence, which we can gain by establishing a few required properties as theorems

$$t \in \text{traces } b \vdash \alpha t \subset \alpha b$$
$$\vdash \text{traces } b = \text{PRFX}(\text{traces } b)$$
$$\vdash \text{finals } b \subset \text{traces } b$$
$$\vdash t \; (\underline{\text{accepts}} \; b) \; e \Rightarrow t \in \text{traces } b$$
$$t \in \text{traces } b ; \; t \neq \langle\rangle \vdash \text{hd } t \; \underline{\text{starts}} \; b$$

Proofs of these theorems depend on the computational induction principle enunciated earlier. In general, since we can only construct behaviours which stop after a finite number of events, we might also expect that the traces could be derived as the prefixes of the finals. The behaviour $(e_1; e_2) \| (e_2; e_1)$ is an obvious counterexample: in some cases of parallel combination, behaviours may disagree with each other. We shall be more interested in behaviours which do terminate at expected points, which we shall call consistent.

$$\text{Consistent} \subset \text{BEHAVIOUR}$$
$$\text{Consistent } b \Leftrightarrow \text{traces } b = \text{PRFX}(\text{finals } b)$$

Consistency is preserved by ; and []. and inconsistency by ; and ||. NULL is not consistent, so we do not expect to use it much in defining other behaviours.

In our set theory, all sets, including BEHAVIOUR, have an equality defined amongst their elements. We can be sure that there are many behaviours in the set, by looking at the functions α, finals, and traces defined over it. For example, $e_1 \neq e_1; e_2$, since their alphabets are different. On the other hand, although our intuition tells us that b and b [] b should be the same behaviour, our theory is not sufficiently developed to prove this. As usual we could repair the defect by introducing axioms to state this and other basic algebraic properties as equalities. We should then have to show that they did not equate behaviours detectably different in their alphabet or traces. The alternative is to define equivalence under these functions, and show that the desired properties are theorems for this relationship. This can be made more attractive technically by borrowing the idea of reduction from Larch, which we express by making BEHAVIOUR isomorphic to its equivalence classes under the functions.

$$\forall b_1, b_2 : \text{BEHAVIOUR}.$$
$$\alpha\, b_1 = \alpha\, b_2 \;\&\; \text{finals } b_1 = \text{finals } b_2 \;\&$$
$$\text{traces } b_1 = \text{traces } b_2 \Rightarrow b_1 = b_2$$

We can now use standard equality theory rather than define a new relation, and take as a natural (semantic) model of BEHAVIOUR a subset of $\mathbb{P}(E) \times \mathbb{P}(seq(E)) \times \mathbb{P}(seq(E))$, defined in part by the interrelationships of the theorems above. The theorems below establish the standard algebraic properties of the operators that we expect to hold under interpretation. such properties are a useful source of conjectures whose establishment gives grounds for confidence in the definitions made They are also sufficient on their own to perform useful proofs.

$$\vdash (b_1; b_2); b_3 = b_1; (b_2; b_3) \quad (C1)$$
$$b; SKIP = SKIP; b = b \quad (C2)$$

$$b \,[]\, b = b \quad (C3)$$
$$b_1 \,[]\, b_2 = b_2 \,[]\, b_1 \quad (C4)$$
$$(b_1 \,[]\, b_2) \,[]\, b_3 = b_1 \,[]\, (b_2 \,[]\, b_3) \quad (C5)$$
$$b \,[]\, NULL = b \quad (C6)$$

$$b \,||\, b = b \quad (C7)$$
$$b_1 \,||\, b_2 = b_2 \,||\, b_1 \quad (C8)$$
$$(b_1 \,||\, b_2) \,||\, b_3 = b_1 \,||\, (b_2 \,||\, b_3) \quad (C9)$$
$$b \,||\, SKIP = SKIP \,||\, b = b \quad (C10)$$

$$b; NULL = b \,||\, NULL \quad (C11)$$
$$(b_1 \,[]\, b_2); b_3 = (b_1; b_3) \,[]\, (b_2; b_3) \quad (C12)$$
$$b_1; (b_2 \,[]\, b_3) = (b_1; b_2) \,[]\, (b_1; b_3) \quad (C13)$$

Since [] and || are both commutative and associative. and have units. we may define

$$[] \in \mathbb{F}(BEHAVIOUR) \rightarrow BEHAVIOUR$$
$$[] = continuation([], NULL)$$

$$|| \in \mathbb{F}(BEHAVIOUR) \rightarrow BEHAVIOUR$$
$$|| = continuation(||, SKIP)$$

where continuation, as defined in appendix A, places the dyadic operator between elements of the set. Another derived operator is ?, defined by

$$? \in \mathrm{BEHAVIOUR} \to \mathrm{BEHAVIOUR}$$
$$?b = b \,[]\, \mathrm{SKIP}$$

The behaviour ?b represents a choice in the world between behaving like b and doing nothing.

It is convenient when describing and reasoning about behaviours to be able to deal with partial definitions, or constraints, as well as complete definitions. If one behaviour constrains another, then we expect it to control the possible orderings of the events which it comprises, whilst leaving any others unconstrained.

$$\underline{sat} \in \mathrm{BEHAVIOUR} \leftrightarrow \mathrm{BEHAVIOUR}$$
$$b_1 \,\underline{sat}\, b_2 \Leftrightarrow \forall t : \mathrm{traces}\, b_1 .\ t \downarrow \alpha b_2 \in \mathrm{traces}\, b_2 \ \&\ \forall t : \mathrm{finals}\, b_1 .\ t \downarrow \alpha b_2 \in \mathrm{finals}\, b_2$$

This clearly has some resemblance to the definition of ||. Intuitively, the behaviour is to be completed by describing the rest of it to run in parallel. Formally,

$$\vdash b_1 \,\underline{sat}\, b_2 \ \&\ \alpha b_2 \subset \alpha b_1 \Leftrightarrow b_1 = b_1 \,||\, b_2 \Leftrightarrow \exists b : \mathrm{BEHAVIOUR} .\ b_1 = b_2 \,||\, b$$

and it follows from the algebraic properties of || that this restriction of $\underline{sat}$ defines a semilattice on BEHAVIOUR, with || as the join operation.

$$\underline{sat}_S = \underline{sat} \cap \{(b, b') \mid b, b' : \mathrm{BEHAVIOUR} \mid \alpha b' \subset \alpha b\}$$

$$\vdash b \ \underline{sat}_S \ b \qquad (S1)$$
$$b_1 \ \underline{sat}_S \ b_2 \ \& \ b_2 \ \underline{sat}_S \ b_1 \Rightarrow b_1 = b_2 \qquad (S2)$$
$$b_1 \ \underline{sat}_S \ b_2 \ \& \ b_2 \ \underline{sat}_S \ b_3 \Rightarrow b_1 \ \underline{sat}_S \ b_3 \qquad (S3)$$
$$b \ \underline{sat}_S \ SKIP \qquad (S4)$$

Theorems S1 and S4 clearly apply also to $\underline{sat}$ The fundamental theorem for reasoning about behaviours follows directly from the definition.

$$\vdash S \subset \alpha b_2 \ \& \ b_1 \ \underline{sat} \ b_2 \Rightarrow$$
$$\{t \downarrow S \mid t : traces \ b_1\} \subset \{t \downarrow S \mid t : traces \ b_2\}$$

The following theorems allow us to establish satisfaction of one behaviour by another in each case, if the alphabet restrictions are met, $\underline{sat}_S$ can be substituted for $\underline{sat}$ and the transitive property used to construct chains of satisfaction.

$$\vdash b_1 \parallel b_2 \ \underline{sat}_S \ b_2 \qquad (S5)$$
$$Independent(b_1, b_2) \Rightarrow b_1 \ \underline{sat} \ b_2 \ \& \ b_2 \ \underline{sat} \ b_1 \qquad (S6)$$
$$Independent(b_1, b) \ \& \ b_1 \ \underline{sat} \ b_2 \Rightarrow$$
$$b_1 \ \underline{sat} \ b_2 \ \Box \ b \qquad (S7)$$
$$b_1 \ \underline{sat} \ b \ \& \ b_2 \ \underline{sat} \ b \Rightarrow b_1 \ \Box \ b_2 \ \underline{sat} \ b \qquad (S8)$$
$$Independent(b_1, b_2) \Rightarrow b_1 ; b_2 \ \underline{sat}_S \ b_1 \parallel b_2 \qquad (S9)$$
$$Independent(b_1, b_4) \ \& \ Independent(b_2, b_3) \ \&$$
$$b_1 \ \underline{sat} \ b_3 \ \& \ b_2 \ \underline{sat} \ b_4 \Rightarrow$$
$$b_1 ; b_2 \ \underline{sat} \ b_3 ; b_4 \qquad (S10)$$

Let us now turn to the effect of the events on the observations. We may imagine stationing an observer where he can see the events as they occur in the part of the world whose behaviour we have described. (Such an observer makes a suitable operator for the information system). He may choose to record various levels of information: he may count the events that occur, he may record which events have occurred, he may record the order in which they occur; or he may even record the times at which they happen. From these basic observations of the world, we may deduce others.

and the richer the information recorded, the more we shall be able to deduce subsequently When we developed the theory of behaviours, we decided to restrict ourselves to specifying only relative orderings in time, with the understanding that we could not express some constraints in that framework We shall take an analogous decision here, and deal only with observations defined by the order of events, which we can formalize as functions in $seq(E) \rightarrow X$ for some X. This is adequate for the IFIP specification presented here, but could not be used, for example, for deriving the interest payable on a savings account A specification of the IFIP problem based on a full treatment of time is given in [Gustafsson81] and makes an interesting contrast with our approach.

Since any trace of a behaviour may occur, the domain of the observation should include them all. Conversely, as no other sequences will be observed, the domain need be no larger than that. We shall define our observations over precisely the observable sequences of events

2.2 Defining the alphabet of events

So far we have treated the events in the system as an arbitrary, unstructured set If we examine a typical information system application, however we find that we have the 'same' event occurring to different members of some set of entities, more correctly, that the events form families whose members are distinguished by the value of one or more parameters For example, in the conference system we have the family of events corresponding to the submission of papers and parameterized by the paper and its author, the family of events for refereeing parameterized by paper and referee, and so on

We should like to use the natural notation $f\ P$, where f is the family and P the parameter value, to denote an event We also need to capture the

intuition that an event is identified uniquely by f and P. We can formalize this directly by giving axiom schemas for this type of term.

$$f \in \mathrm{FAMILY} \Rightarrow f\ P \in E$$
$$f_1, f_2 \in \mathrm{FAMILY} \ \&\ f_1P_1 = f_2P_2 \Rightarrow f_1 = f_2 \ \&\ P_1 = P_2$$

In general, we expect the parameter terms to be tuples, so that we see events of the form Referee(p, r). Terms of the form f or P are to be taken as parameters of the schema.

This extension of the axiom schemas of our set theory to enable us to reason with a greater variety of terms can be compared with the 'abstract syntax' approach of [Sufrin83]. In that treatment, each f in FAMILY would be associated with an injection from its parameter set into E, and the disjointness of the ranges of the injections asserted. To lighten the load of stating these properties, a new syntactic form is introduced, and the associated proof rules given informally. The proof rules for events we have introduced above are then theorems of the construction. We can complete the 'isomorphism' between the two approaches by defining injections from the parameter sets for each family

$$\mathrm{rep} \in \mathrm{FAMILY} \rightarrow (X \rightarrowtail E)$$
$$(\mathrm{rep}\ f)\ x = f\ x$$

This can be extended to give the set of events corresponding to a set of parameters in a family

$$\underline{\mathrm{from}} \in \mathrm{FAMILY} \times \mathbb{P}(X) \rightarrow \mathbb{P}(E)$$
$$f\ \underline{\mathrm{from}}\ S = \{f\ s \mid s : S\}$$

so we can talk of Submit <u>from</u> (P×A) and so on.

Once we have defined the behaviour of our world, we know the possible sequences of events which may be observed to happen in it. A corresponding sequence of events should be entered into the system which imitates the world. We can make the system reject sequences of events which could not possibly happen, but we cannot prevent the entry of sequences which might have happened but didn't. We can make provision for recovery from the simplest form of error – accidentally entering the wrong event – by providing a new event, OOPS, with the intention (that we shall make formal later) that entering OOPS 'undoes' the effect of the previous event. This new event is distinct from the events in the world.

$$\mathrm{OOPS} \notin E$$
$$E_{OOPS} = E \cup \{\mathrm{OOPS}\}$$

A system with capacity for undoing erroneously entered events must accept sequences of events from the set E_{OOPS}. Given such a sequence, we can define the corresponding sequence of desired events, where the erroneous entries have been cancelled.

$$\mathrm{edit} \in \mathrm{seq}(E_{OOPS}) \rightarrow \mathrm{seq}(E)$$

$$\mathrm{edit}(\langle\rangle) = \langle\rangle$$
$$\mathrm{edit}(\langle \mathrm{OOPS} \rangle) = \langle\rangle$$
$$\forall t : \mathrm{seq}(E_{OOPS}) \cdot$$
$$e \in E \Rightarrow \mathrm{edit}(t^\frown\langle e\rangle) = \mathrm{edit}(t)^\frown\langle e\rangle \;\&$$
$$e \in E_{OOPS} \Rightarrow \mathrm{edit}(t^\frown\langle e, \mathrm{OOPS}\rangle) = \mathrm{edit}(t)$$

The rather unconventional right-recursive definition is much more convenient in definition and proof than the left-recursive form. Examination of the definition will show that we have defined the behaviour in cases left

unmentioned in the informal description two consecutive OOPSes cancel, leaving the event before 'done', and an initial OOPS will be ignored. The former is a reasonably natural choice (and certainly more efficiently implementable), while the latter is made for the technical convenience of having edit total. As one would expect, edit leaves a sequence of events not containing OOPS unchanged. (The proof is by induction, trivially).

$$\vdash \forall t: \mathrm{seq}(E).\ \mathrm{edit}(t) = t$$

We should like to characterize the set of sequences of events with corrections which should be accepted by a system with given behaviour. Any such sequence, when edited, must give a legitimate trace of the behaviour. The same property must also hold for every prefix so that impossible events are rejected immediately.

$$\mathrm{backtracks} \in \mathrm{BEHAVIOUR} \to \mathbb{P}(\mathrm{seq}(E_{\mathrm{OOPS}}))$$
$$\mathrm{backtracks}(b)(t) \Leftrightarrow$$
$$\quad \forall t': \mathrm{PRFX}(t).\ \mathrm{edit}(t') \in \mathrm{traces}(b)$$

There are certain properties that we expect of backtracks. First, where an operator makes no mistakes, the original sequence of events should be valid, that is, every member of traces(b) is in backtracks(b). Second, OOPS should always be allowed as an event. Finally, as events are entered one at a time, we expect the set to be prefix closed. It might be possible to use these properties to define backtracks. Instead, we must establish them as theorems and thus justify our more constructive approach. The first is trivial, given the theorem on edit and the prefix closure of traces; the second follows from case analysis on the end of the trace, and the third is proved by induction.

$$b \in \mathrm{BEHAVIOUR}$$
$$\quad \vdash \mathrm{traces}(b) \subseteq \mathrm{backtracks}(b)$$
$$\quad \vdash \forall t: \mathrm{backtracks}(b).\ t^\frown\langle \mathrm{OOPS} \rangle \in \mathrm{backtracks}(b)$$
$$\quad \vdash \forall t: \mathrm{backtracks}(b).$$
$$\qquad \forall t': \mathrm{PRFX}(t).\ t' \in \mathrm{backtracks}(b)$$

The observations on the system must now be defined over backtracks(b), rather than traces(b). It is possible to define functions whose results depend on the presence of OOPS events: we could, for instance, just count the number of errors made by the operator of the system. On the other hand, we expect the observations for which the system was designed to depend for their results only on the events correctly entered. We can characterize such functions as *forgiving*.

$$\begin{aligned}&\text{Forgiving}(S) = \\ &\quad \{r : \text{seq}(E_{OOPS}) \leftrightarrow S \mid \\ &\quad\ \forall t_1, t_2 : \text{seq}(E_{OOPS});\ s : S \cdot \\ &\quad\quad \text{edit}(t_1) = \text{edit}(t_2) \Rightarrow r(t_1, s) \Leftrightarrow r(t_2, s)\}\end{aligned}$$

Given a sequence of events within the backtracks of a behaviour, we expect the possible next events to be those acceptable to the edited trace within traces(b). If we define a new relation to include OOPS

$$\begin{aligned}&\underline{\text{accepts}}_B \in \text{BEHAVIOUR} \rightarrow (\text{seq}(E_{OOPS}) \leftrightarrow E_{OOPS}) \\ &t\ (\underline{\text{accepts}}_B\ b)\ e \Leftrightarrow t^\frown\langle e\rangle \in \text{backtracks}(b)\end{aligned}$$

we can state this as

$$b \in \text{BEHAVIOUR} \vdash \text{Forgiving}(E_{OOPS})(\underline{\text{accepts}}_B\ b)$$

The proof follows from the definitions of the terms and case analysis on the accepted event.

The function edit defines equivalence classes over $\text{seq}(E_{OOPS})$, each of which contains a unique member of seq(E). We can systematically define relations on the enhanced traces from those on the traces of events in the world by making use of this.

$$\begin{aligned}&\text{enhancement} \in (\text{seq}(E) \leftrightarrow X) \rightarrow (\text{seq}(E_{OOPS}) \leftrightarrow X) \\ &\text{enhancement}(r) = \text{edit}; r\end{aligned}$$

Clearly, by construction, since forgivingness requires only that the equivalence classes of edit be respected by a relation,

$$r \in (\mathrm{seq}(E) \leftrightarrow S) \vdash \mathrm{Forgiving}(S)(\mathrm{enhancement}(r))$$

We can therefore define observations for the system in the natural way over the sequences of real events which comprise the traces, and characterize the functions over the backtracks that we actually require by enhancement

3 The specification

We have now completed the theoretical background, and have some idea of the shape the system will have. To make this pattern more specific, we shall adopt the image of an 'intelligent filing cabinet', which allows the recording of particular events as they occur, and the retrieval of information for guiding the activities of the committees, but does not attempt to automate any of their activities.

3.1 The behaviour of the system

We should begin by giving the behaviour a name. This provides a new constant in our theory whose properties we may then state

$$CRIS \in BEHAVIOUR$$

We have developed mechanisms for constructing a behaviour from its subcomponents. A natural division in the informal description is that between the programme committee, who are responsible for the papers submitted, and the organizing committee, who must take care of the delegates. We shall begin with the events concerning the papers once they have arrived.

A paper comes into the system by being submitted by its author. It may then be sent for consideration by one or more referees. Normally, this will result in a verdict being returned, but in exceptional circumstances we shall allow the refereeing process to be truncated at the request of one or other party. When all the referees' reports are received, a decision can be made and the author informed. An accepted paper must be assigned to a conference session. We also provide for the possible acknowledgement of the receipt of a paper.

To formalize this, let us represent the papers by the set P, referees by the set R, their verdicts by V, authors by A, and times for presentation by T. We shall assume these (and the rest of the sets introduced) to be finite, so that all our combinators on behaviours can be used freely. As usual, lower case single letter identifiers and their variants belong to the upper case sets. We shall allow for two possible decisions, acceptance and refusal

$$\{\text{Submit, Referee, Report, Cancel, Decide, Inform, Schedule, Acknowledge}\} \subset \text{FAMILY}$$

$$\text{Dec} = \{\text{ACCEPTED, REJECTED}\}$$

From the viewpoint of the referee, some of the papers will arrive, and he will either referee them or return them. If he does neither, then he may be told not to bother. He cannot referee the same paper twice.

REF

$$\text{Referee}(r, p);\ (\Box\{\text{Report}(r, p, v) \mid v: V\}) \,\square\, \text{Cancel}(r, p)$$

$$\forall r: R;\ p: P.\ \text{CRIS} \underline{\text{sat}}_S\ ?\text{REF}$$

If this is the only restriction on refereeing, then an arbitrary number of papers may be sent to any one referee. This is in accord with our declared policy of leaving decisions to the users: we shall expect to make the workload of the referees available to the programme committee to help them make a selection. (They should also be aware of other constraints beyond the scope of this system, such as, for example, a referee being a spouse of the author)

Note that there is a difference between undoing the selection of a referee by using OOPS and cancelling the appointment with Cancel. In the latter

case. the referee can not subsequently be reappointed to the paper (as one might expect) Cancellation can also take place at any time after the appointment. whereas the use of OOPS applies only to the immediately preceding event. whatever that may be

The author. having submitted a paper. may get an acknowledgement. and will be notified of the programme committee's decision.

AUTHORP

```
?Acknowledge(p, a);
[] {Inform(p, a, d) | d: Dec}
```

$\forall p: P;\ a: A.\ CRIS\ \underline{sat}_S\ ?AUTHORP$

We can then describe how the programme committee must handle a paper They may choose to acknowledge the receipt of a paper at least until a decision is reached Subsequently, they must inform the author and schedule the paper if necessary. where scheduling requires the choice of a time slot for the paper to be presented

DEC

```
  Decide(p, ACCEPTED);
  (Inform(p, a, ACCEPTED) ||
   [] {Schedule(p, t) | t: T})
[] Decide(p, REJECTED); Inform(p, a, REJECTED)
```

PAPER

```
[](((Submit(p, a);
    ?Acknowledge(p, a) || (||{?REF | r: R});
    DEC)
          | a: A)
```

$\forall p: P.$ CRIS $\underline{sat}_S$?PAPER

We can establish that

$\vdash \forall p: P;\ r: R.$ PAPER $\underline{sat}_S$?REF
$\vdash \forall a: A;\ p: P.$ PAPER $\underline{sat}_S$ AUTHORP

by using the satisfaction theorems of the previous chapter as lemmas. Hence, given the transitivity of $\underline{sat}_S$, we can be sure that any system which satisfies ?PAPER will meet the rest of the specification so far.

In scheduling the papers, we have one more, rather unexpected, 'actor' in our behaviour: the conference time slots. These must be allocated to at most one paper.

$\forall t: T.$ CRIS $\underline{sat}_S$?[]{Schedule(p, t) | p: P}

The programme committee may also take suggestions for potential authors, issue a call for papers, and then register letters of intent. We shall assume that these are not mandatory, and that they will not happen once a paper has actually been submitted

$\{Suggest_A, Call, Intend\} \subset FAMILY$

PRESUBMIT

```
?(Suggest_A(a); Call(a)); ?Intention(a)
```

$\forall p: P;\ a: A.\ CRIS\ \underline{sat}_S\ ?(PRESUBMIT;\ AUTHORP)$

The organizing committee must coordinate the invitations to delegates. These delegates may be suggested, in which case they will certainly be sent an invitation. They may then accept the invitation, and will receive a reply from the organizers. We shall also deal with the self-invited, returning the answer 'no'. Formally, we use D for the set of delegates.

$\{Suggest_D, Invite, Accept, Reply\} \subset FAMILY$

DEL

```
Invite(d);
?(Accept(d); []{Reply(d, d') | d': Decision})
```

DELEGATE

```
  (Suggest_D(d); DEL)
[] DEL
[] (Accept(d); Reply(d, REJECTED))
```

$\forall d: D.\ CRIS\ \underline{sat}_S\ ?DELEGATE$

Authors, as we have established, submit papers and expect replies from the programme committee. They are also entitled to invitations as a result of submitting a paper. Their complete behaviour as far as the system is concerned is thus

AUTHOR

```
PRESUBMIT; Submit(p, a); (DEL[a/d] || AUTHORP)
```

$\forall p: P;\ a: A.\ \text{CRIS}\ \underline{\text{sat}}_S\ ?\text{AUTHOR}$

We can readily show that

$\vdash \text{AUTHOR}\ \underline{\text{sat}}_S\ \text{PRESUBMIT; AUTHORP}$

and hence that satisfaction of the requirements of AUTHOR guarantees satisfaction of those of AUTHORP.

We have now defined the behaviour of the world as seen from various viewpoints. We want all these activities to go on, as far as possible, in parallel we can also say at this stage that there are no other events. We can therefore define CRIS as the parallel combination of all the behaviours it must satisfy

$$\begin{aligned}\text{CRIS} = \ & ||\{?\text{PAPER} \mid p: P\} \ ||\ ||\{?\text{DELEGATE} \mid d: D\} \ ||\\ & ||\{?\text{AUTHOR} \mid p: P;\ a: A\} \ ||\\ & ||\{?\ \square\ \{\text{Schedule}(p, t) \mid p: P\} \mid t: T\}\end{aligned}$$

3.2 The observations

Having defined the behaviour in which we are interested, we can now discuss some observations that we should like to make. It is convenient to name the results of applying some of the functions on behaviours to CRIS.

$$\begin{aligned} \mathrm{TRACES} &= \mathrm{traces}(\mathrm{CRIS}) \\ \mathrm{BTRACKS} &= \mathrm{backtracks}(\mathrm{CRIS}) \\ \underline{\mathrm{accepts}} &= \underline{\mathrm{accepts}}_{B}(\mathrm{CRIS}) \end{aligned}$$

We intend to define some functions from BTRACKS to provide the substitute observations which the user needs, but we must first decide what observations are required. Our ultimate authority is the client, but as usual we should like some pattern for shaping our dialogue with him.

The observations we can define are bounded above by the information which BTRACKS conveys. The restrictions implied by the theory itself were discussed as it was developed: by choosing specific events associated with particular information we have limited ourselves further. It is no use trying to ask about consequences of events which we have decided not to record, and if we find that we need to do so, we must revise our ideas of the events in the world. Completeness is confirmed by exhibiting a definition of the observation.

Let us consider the observations that the user might want to make of the state of the refereeing. These might include the referees still to produce reports, the papers with reports outstanding, papers which need referees appointed, the referees' reports on a paper, and so on, and indeed, the actual observations chosen may be constrained by aspects of the design not yet settled. For example, if a referee's verdict is expressed on a scale out of ten, then we may conveniently show all the verdicts for a particular paper in one screenful, whereas if they are more involved they may need to be taken one at a time. At this level of abstraction of specification,

therefore, we should aim to find a small set of observations from which we can expect to derive others, and delay the determination of the actual set to be presented to the user until later

There are two properties of CRIS that help us in this. In general, the observations define the effect, or "meaning", of a sequence of events. Our intuition in this case, however, is that individual (real) events have a meaning independent of the rest of the sequence in which they appear (although these sequences are constrained by the behaviour to be meaningful collectively). For example, Submit(p, a) means that paper p has been received from author a, and it occurs only in sequences where no other author has submitted the paper It therefore seems that knowing the set of events which has occurred might be enough to deduce any further interesting observations.

Further support for this idea comes from the second property of CRIS, that events happen once if at all: to know that they have occurred is to know how often they have occurred. Further, if the behaviour tells us that it is one of a sequence of events, then we know the order in which those events occurred, and if it is part of a set of independent alternatives, that the events of the other alternatives will not occur. Only where events may occur in parallel is there any doubt as to the actual order of occurrence, but parallel combination in CRIS is used only with independent threads of behaviour we are unlikely to want to know if a given paper was received before some person was invited to the conference. Thus, given an observation from which we can determine if a given event has occurred, we expect to be able to define almost any (forgiving) function definable from the traces themselves, the remainder being 'uninteresting'.

For sequences of real events, this observation can be just the alphabet of the trace For the more general case of traces with corrections, we have

$$obs \in BTRACKS \rightarrow \mathbb{P}(E)$$
$$obs = enhancement(\alpha)$$

$$\vdash \text{ran obs} = \{\alpha\ t \mid t : \text{TRACES}\}$$

which, applying the theorem of alphabets of behaviours, gives

$$\vdash \text{ran obs} \subset \mathbb{P}(\alpha\ \text{CRIS})$$

This observation is rather inconveniently monolithic, and we can consider breaking it up into parts that, taken together, provide the same information: that is, if we have f_1 in $\mathbb{P}(E) \rightarrow X_1$, f_2 in $\mathbb{P}(E) \rightarrow X_2$, and so on, then the combined function $f_1, f_2, \ldots$ defines an isomorphism between ran obs and the range of the combination.

Our programmer's intuition says that this can 'obviously' and 'naturally' be done by providing one observation for each family, which for Suggest_A would yield the set of potential authors suggested to the committee, for Referee the relationship between referees and papers, and so on. With the suspicion of words in inverted commas born of experience, we should like a little more justification for this proposal, particularly as different programmers may choose different interpretations of the observations: for example, by returning the referees who are still to report on a paper rather than all the referees who have ever been asked to report on it.

The justification is a construction of the isomorphism between the unstructured and the structured observations. We have already established by axiom the isomorphism between the parameter values of a family and the set of events they define. Projection of the set of events which have occurred onto a single family is clearly not injective: if, however, we combine projections so that every event is represented somewhere in the image, then the resulting operation will be. This shows that one observation per family is necessary, and that it is sufficient to consider only the events in the alphabet of the behaviour, as defined by the parameters of the families.

This simple process of projection and label stripping uniformly gives us the historical rather than the current observation of processes like refereeing: justifying the current view is more involved, and we shall return to it when we consider implementations in the next chapter. In the meantime, let us define a pattern for the observations for a given family and parameter set

$$\text{Observation} \in \text{FAMILY} \times \mathbb{P}(X) \rightarrow (\text{TRACES} \rightarrow \mathbb{P}(X))$$
$$x \in \text{Observation}(f, S)(t) \Leftrightarrow x \in S \;\&\; f\,x \in \alpha t$$

$$\begin{aligned}
\text{Submitted} &= \text{Observation}(\text{Submit}, P \times A)\\
\text{Refereeing} &= \text{Observation}(\text{Referee}, P \times R)\\
\text{Cancelled} &= \text{Observation}(\text{Cancel}, P \times R)\\
\text{Reports} &= \text{Observation}(\text{Report}, (P \times R) \times V)\\
\text{Acknowledged} &= \text{Observation}(\text{Acknowledge}, P \times A)\\
\text{Decided} &= \text{Observation}(\text{Decide}, P \times \text{Dec})\\
\text{Informed} &= \text{Observation}(\text{Inform}, P \times (A \times \text{Dec}))\\
\text{Scheduled} &= \text{Observation}(\text{Schedule}, T \times P)\\
\text{Suggested}_A &= \text{Observation}(\text{Suggest}_A, A)\\
\text{Called} &= \text{Observation}(\text{Call}, A)\\
\text{Intending} &= \text{Observation}(\text{Intend}, A)\\
\text{Suggested}_D &= \text{Observation}(\text{Suggest}_D, D)\\
\text{Invited} &= \text{Observation}(\text{Invite}, D)\\
\text{Accepted} &= \text{Observation}(\text{Accept}, D)\\
\text{Replied} &= \text{Observation}(\text{Reply}, D \times \text{Dec})
\end{aligned}$$

4 Implementing the system

The functional specification of the system was completed in the previous chapter: by the end, we have a description of what the system is to do, but with no indication of how, even to the extent of indicating hardware or user interface. At this stage, the project should be reviewed. The client can decide how much a system with the specified capabilities is worth, and this limits the budget of the implementation phase. The designer can then decide if the system can be provided within the budget, possibly by exploring the outlines of some designs in private. If both parties are then satisfied, the contracts can be signed and the implementation begun.

4.1 The properties of implementations

In this section, we shall establish a framework for talking about a particular implementation by developing the properties that all implementations must have.

We can regard the implemented system as a 'black box', which is provided with means for indicating the occurrence of individual events (including OOPS), and for determining the current values of the observations at any time. The box presumably has internal state, since the observations depend on the sequence of events input, in a way which mimics the observations defined for the sequences of events in the specification. An attempt to enter an event which is not acceptable as a continuation to the previous sequence causes no change in the observations. We can infer that the state must be rich enough to support the computation of the results of the observations and the effects of the operations which follow the indication of an event.

Formally, let us assume that the internal states are in some set DB and denote members of it by identifiers like d. To allow the results of the observations to be computed, the implementation must provide a function

$$obs_M \in DB \rightarrow ran\ obs$$

where the partiality of the function allows for 'unused' states. This reminds that the implementation must provide at least as many states as there are distinct observations.

The effect of entering an event in a given state may be formalized as a function

$$effect \in E_{OOPS} \rightarrow (DB \rightarrow DB)$$

and this can be extended naturally to sequences of events

$$effect \in seq(E_{OOPS}) \rightarrow (DB \rightarrow DB)$$
$$effect = extension(effect)$$

where extension is defined in appendix A. If the implementation is to mimic the specification, then, for some InitDB in DB, we must have

$$\forall t: BTRACKS.\ obs\ t = obs_M(effect\ t\ InitDB)$$

If, however, we want to interpret t in this axiom as "the sequence of events entered by the user", then this requirement is not strong enough: the result of any sequence must be correct. Our goal for establishing the correctness of an implementation then becomes

$$C \triangleq \forall t: seq(E_{OOPS}).\ obs\ t = obs_M(effect\ t\ InitDB)$$

The function obs is so far defined only over BTRACKS. and we should formalize what we mean by observing arbitrary sequences.

$$\forall t: \overline{BTRACKS} .\ obs\ t = obs\ (purify\ t)$$

where

$$purify \in seq(E_{OOPS}) \twoheadrightarrow BTRACKS$$
$$purify(\langle\rangle) = \langle\rangle$$
$$purify\ s\ \underline{accepts}\ e \Rightarrow$$
$$\quad purify(s * \langle e\rangle) = purify(s) * \langle e\rangle$$
$$\neg purify\ s\ \underline{accepts}\ e \Rightarrow purify(s * \langle e\rangle) = purify(s)$$

The interpretation of purify clearly satisfies our informally stated requirement of no observable change caused by illegal events. Purification is clearly idempotent, and hence an identity on BTRACKS: hence

$$\vdash \forall t: seq(E_{OOPS}) .\ obs\ t = obs(purify\ t)$$

The obvious strategy for proving that G holds for an implementation is by induction. In expanding out the basis step, we arrive at the goal

$$G1 \triangleq obs\ \langle\rangle = obs_M\ InitDB$$

and for the inductive step

$$G2 \triangleq obs\ t = obs_M\ (effect\ t\ InitDB) \vdash$$
$$obs(t*\langle e\rangle) = obs_M(effect\ (t*\langle e\rangle)\ InitDB)$$

This latter goal can be subdivided according to the validity of the event introduced to give

$$G3 \triangleq obs\ t = obs_M(effect\ t\ InitDB);\ purify\ t\ \underline{accepts}\ e \vdash obs(t^*\langle e\rangle) = obs_M(effect\ (t^*\langle e\rangle)\ InitDB)$$

(a particular form of G2), and

$$G4 \triangleq obs\ t = obs_M(effect\ t\ InitDB);\ \neg purify\ t\ \underline{accepts}\ e \vdash obs_M(effect\ t\ InitDB) = obs_M(effect\ (t^*\langle e\rangle)\ InitDB)$$

which together subsume G2 (the latter restating what we have always known, that illegal events have no effect). If we assume the forgiveness of obs, then we have

$$Forgiving(obs) \vdash \forall t: BTRACKS.\ obs\ t = obs(edit\ t)$$

and hence

$$Forgiving(obs) \vdash \forall t: seq(E_{OOPS}).\ obs(purify\ t) = obs(edit(purify\ t)) = obs\ t$$

which yields new subgoals from G3 by analysis of the last event

$$G5 \triangleq obs\ t = obs_M(effect\ t\ InitDB);\ purify\ t\ \underline{accepts}\ e;\ e \in E \vdash obs(t^*\langle e\rangle) = obs_M(effect\ (t^*\langle e\rangle)\ InitDB)$$

$$G6 \triangleq obs_M(effect\ OOPS\ InitDB) = obs_M\ InitDB$$

$$G7 \triangleq \text{obs}\ t = \text{obs}_M(\text{effect}\ t\ \text{InitDB});\ \text{purify}\ t\ \underline{\text{accepts}}\ e \vdash \text{obs}_M(\text{effect}\ \text{OOPS}\ (\text{effect}\ (t^*\langle e\rangle)\ \text{InitDB})) = \text{obs}_M(\text{effect}\ t\ \text{InitDB})$$

Of the goals so far, only G1 and G5 depend on the form of obs. By adopting the definition of the previous section, we may rewrite these as

$$G8 \triangleq \text{obs}_M\ \text{InitDB} = \Phi_E$$

$$G9 \triangleq \text{obs}\ t = \text{obs}_M(\text{effect}\ t\ \text{InitDB});\ \text{purify}\ t\ \underline{\text{accepts}}\ e\ ;e \in E \vdash \text{obs}_M(\text{effect}\ e\ (\text{effect}\ t\ \text{InitDB})) = \text{obs}_M(\text{effect}\ t\ \text{InitDB}) \cup \{e\}$$

respectively. The goal set to be established after this analysis is {G4, G6, G7, G8, G9}.

4.2 A particular implementation

Inspection of the goals shows that, depending on the conditions satisfied by the previous history and the event, the observable effect to be produced by the event is no change (G4), the result before the previous event (G7), or is derived from the previous observation by adding an event (G9), a process which is guaranteed in the case of CRIS to produce a change in the observation. We might therefore take as the state a pair of sets of events

$$DB = \mathbb{P}(E) \times \mathbb{P}(E)$$

where one represents the current and the other the previous observation. The observation function is trivial

$$\text{obs}_M(c, p) = c$$

Formally, we can take G3 and G4 and our new found (although not unexpected) knowledge that events always have an effect, and deduce that

$$G3;\ G4 \vdash \forall t_1, t_2 : seq(E_{OOPS});\ e : E.$$
$$effect\ t_1\ InitDB = effect\ t_2\ InitDB \Rightarrow$$
$$purify\ t_1\ \underline{accepts}\ e \Leftrightarrow purify\ t_2\ \underline{accepts}\ e$$

Informally, we can interpret this as the discovery that, if the system is to perform as specified, then the states of the implementation must contain enough information to decide when events are allowed. OOPS, of course, is always possible. An implementation must have this property to have any chance of correctness.

If we take the stronger condition of equality of the observation implying equivalent acceptance, we may rewrite the precondition on correctness as

$$G11 \triangleq \forall t_1, t_2 : TRACES;\ e : E \mid \alpha\, t_1 = \alpha\, t_2.$$
$$t_1\ \underline{accepts}\ e \Leftrightarrow t_2\ \underline{accepts}\ e$$

We should like to establish this for CRIS without having to enumerate all its traces. We can generalize the property to all behaviours

$$DBLike \sqsubset BEHAVIOUR$$
$$DBLike(b) \Leftrightarrow \forall t_1, t_2 : traces(b);\ e : E \mid \alpha\, t_1 = \alpha\, t_2.$$
$$t_1\ (\underline{accepts}\ b)\ e \Leftrightarrow t_2\ (\underline{accepts}\ b)\ e\ \&$$
$$finals\ b\ t_1 \Leftrightarrow finals\ b\ t_2$$

The clause involving `finals` strengthens the condition of G11 to include the 'event' of termination. It is required of the first behaviour in a sequential combination to preserve the weaker condition, and is preserved by the constructors in a similar way. It is therefore technically convenient to fold the two conditions together when stating the theorems below.

Definitions of properties of behaviours in terms of their sets of traces are not very helpful, since these sets can be large. It is possible to deduce some sufficient (but not necessary) conditions on the structure and alphabet of behaviours which are more readily determined.

$$\vdash \mathrm{DBLike}(\mathrm{SKIP})$$
$$\vdash \mathrm{DBLike}(e)$$
$$b_1, b_2 \in \mathrm{BEHAVIOUR};\ \mathrm{DBLike}(b_1);\ \mathrm{DBLike}(b_2)$$
$$\quad \vdash \mathrm{Independent}(b_1, b_2) \Rightarrow \mathrm{DBLike}(b_1 ; b_2)$$
$$\quad \vdash \mathrm{Separable}(b_1, b_2) \Rightarrow \mathrm{DBLike}(b_1 \,[]\, b_2)$$
$$\quad \vdash \mathrm{DBLike}(b_1 \,||\, b_2)$$

where

$$\mathrm{Separable} \in \mathrm{BEHAVIOUR} \leftrightarrow \mathrm{BEHAVIOUR}$$
$$\mathrm{Separable}(b_1, b_2) \Leftrightarrow$$
$$\quad \forall t_1, t_2 : \mathrm{traces}(b_1 \,[]\, b_2) \mid \alpha t_1 = \alpha t_2 .$$
$$\quad\quad (t_1 \in \mathrm{traces}\, b_1 \Leftrightarrow t_2 \in \mathrm{traces}\, b_1) \ \&$$
$$\quad\quad (t_1 \in \mathrm{traces}\, b_2 \Leftrightarrow t_2 \in \mathrm{traces}\, b_2)$$

Separability is not, of course, expressed as a structural condition, but we shall show two forms of separable behaviour in the next section. In each case above, the preconditions make the proof trivial, and are relatively strong: independence of behaviours joined by ; implies that each event (as in CRIS) occurs only once. We can use these theorems to show

$$\vdash \mathrm{DBLike}(\mathrm{CRIS})$$
$$\vdash \mathrm{G11}$$

Now we have shown the adequacy of the state, let us complete the construction by building the operations upon it, and determining the initial state. Let us split the effect functions along the lines suggested by the conditions of G4 and G9.

$$\forall e{:}E.\ \mathrm{effect}\ e = \mathrm{Id} \oplus \mathrm{effect}_N\ e\ \&\ \mathrm{dom}\ \mathrm{effect}_N\ e = \{\mathrm{effect}\ t\ \mathrm{InitDB} \mid t{:}\ \mathrm{seq}(E_{OOPS}) \mid \mathrm{purify}\ t\ \underline{\mathrm{accepts}}\ e\}$$

With this construction, clearly,

$$\forall t{:}\ \mathrm{seq}(E_{OOPS}).\ \mathrm{effect}\ t = \mathrm{effect}\ (\mathrm{purify}\ t)$$

Then, as a consequence of G11, we have

$$\vdash \forall t{:}\ \mathrm{seq}(E_{OOPS});\ e{:}E.\ \mathrm{effect}\ t\ \mathrm{InitDB} \in \mathrm{dom}\ \mathrm{effect}_N\ e \Leftrightarrow \mathrm{purify}\ t\ \underline{\mathrm{accepts}}\ e$$

which immediately establishes G4. If we choose

$$\forall (c, p){:}\ \mathrm{dom}(\mathrm{effect}_N\ e).\ \mathrm{effect}_N\ e\ (c, p) \in \{(c \cup \{e\}, S) \mid S{:}\ \mathbb{P}(E)\}$$

then G9 is also established, and we can take

$$\mathrm{InitDB} \in \{(\Phi_E, S) \mid S{:}\ \mathbb{P}(E)\}$$

to satisfy G8. Consideration of the requirements of OOPS dictates that these be strengthened to

$$\forall (c, p){:}\ \mathrm{dom}(\mathrm{effect}_N\ e).\ \mathrm{effect}_N\ e\ (c, p) = (c \cup \{e\}, c)$$
$$\mathrm{InitDB} = (\Phi_E, \Phi_E)$$

with

$$\mathrm{effect}\ \mathrm{OOPS}\ (c, p) = (p, c)$$

given which, the remaining goals G6 and G7 are readily established.

4.3 A more efficient implementation

There are three areas in which the efficiency of the proposed implementation is dubious, particularly with regard to the space required: the representation of the previous state for OOPS, the form in which the domains of the operations are expressed, and the representation of the state as an arbitrary set of events. We shall deal with these in turn.

We know that each event occurs at most once, and hence the effect of a legal, real event is to add precisely that event to the observation. This suggests that we can represent one state by the single event needed to derive it from the other: it is convenient to derive the previous state from the current state.

$$DB_1 = \mathbb{P}(E) \times E_{OOPS}$$

We can write a retrieve function to the previous state as

$$\begin{aligned}
&ret \in DB_1 \rightarrowtail\!\!\!\!\!\!\rightarrow \{effect\ t\ InitDB \mid t : TRACES\} \\
&e \in c \Rightarrow ret(c, e) = (c, c - \{e\}) \\
&e \notin c \ \&\ e \neq OOPS \Rightarrow ret(c, e) = (c, c \cup \{e\}) \\
&ret(c, OOPS) = (c, c)
\end{aligned}$$

and, since this is a bijection, immediately derive the corresponding operations from the previous implementation by defining

$$\begin{aligned}
&effect \in E_{OOPS} \rightarrow (DB_1 \rightarrow DB_1) \\
&effect_N \in E_{OOPS} \rightarrow (DB_1 \rightarrow DB_1) \\
&effect\ e = ret;\ effect\ e;\ ret^{-1}
\end{aligned}$$

This gives

$$\begin{aligned}
&e \in E \Rightarrow effect_N\ e\ (c, e') = (c \cup \{e\}, e) \\
&effect\ OOPS\ (c, OOPS) = (c, OOPS)
\end{aligned}$$

$e \in E;\ e \notin c \Rightarrow \text{effect OOPS } (c, e) = (c \cup \{e\}, e)$
$e \in E;\ e \in c \Rightarrow \text{effect OOPS } (c, e) = (c - \{e\}, e)$
$\text{InitDB} = (\Phi_E, \text{OOPS})$

Now let us turn our attention to the domain conditions for the operations We can reshape the collected domain conditions as a relation analogous to the acceptance relations on sequences.

$\underline{\text{accepts}} \in \text{DB} \leftrightarrow E$
$d \ \underline{\text{accepts}}\ e \Leftrightarrow d \in \text{dom}(\text{effect}_N\ e)$

What we are seeking is a relation $\underline{\text{accepts}}_M$ equivalent to this, but expressed in a form more oriented to computation than a set of pairs of states and events Equivalence in this context means giving the same result over the subset of DB reached by sequences of events

$\underline{\text{accepts}}_M \in \text{DB} \leftrightarrow E$
$\forall t: \text{seq}(E_{OOPS});\ e: E.\ \text{effect } t\ \text{InitDB } \underline{\text{accepts}}\ e \Leftrightarrow \text{effect } t\ \text{InitDB } \underline{\text{accepts}}_M\ e$

Making use of the results on accepts and purify, this can be reduced to an equivalent goal

$G12 \triangleq \forall t: \text{BTRACKS};\ e: E.\ \text{effect } t\ \text{InitDB } \underline{\text{accepts}}_M \Leftrightarrow t\ \underline{\text{accepts}}\ e$

and if we agree to construct $\underline{\text{accept}}_M$ as

$\underline{\text{accepts}}_A \in \mathbb{P}(E) \leftrightarrow E$
$\underline{\text{accepts}}_M = \text{obs}_M;\ \underline{\text{accepts}}_A$

then this finally reduces to showing

$G13 \triangleq \forall t: \text{TRACES};\ e: E.\ \alpha t\ \underline{\text{accepts}}_A\ e \Leftrightarrow t\ \underline{\text{accepts}}\ e$

G11 gives us grounds for thinking that such a relation may exist

Let us look at a particular event. Reports(r, p, v). Looking at the specification of the behaviour, it seems that the view REF gives us sufficient context to determine when the event can occur after Referee(r, p), but not if any Report(r, p, v') or Cancel(r, p) have occurred. We should attempt to formalize these ideas and justify our intuition. The basic test of a state looks at the events which have occurred and those which have not.

$$FPAIR = \mathbb{P}(E) \times \mathbb{P}(E)$$

$$\underline{sel} \in FPAIR \leftrightarrow \mathbb{P}(E)$$
$$(o, n)\ \underline{sel}\ S \Leftrightarrow o \subset S\ \&\ Disjoint(n, S)$$

We can take a set of these, and define the new test as the disjunction of the tests of its elements.

$$FILTER = \mathbb{P}(FPAIR)$$

$$\underline{sel} \in FILTER \leftrightarrow \mathbb{P}(E)$$
$$F\ \underline{sel}\ S \Leftrightarrow \exists f : F.\ f\ \underline{sel}\ S$$

and, given a filter for each event

$$\underline{has\ filter} \in BEHAVIOUR \rightarrow (E \leftrightarrow FILTER)$$

we can construct, given some filter F for e

$$e\ (\underline{has\ filter}\ CRIS)\ F \Rightarrow S\ \underline{accepts}_A\ e \Leftrightarrow F\ \underline{sel}\ S$$

and hence can define in general

$$e\ (\underline{has\ filter}\ b)\ F \Leftrightarrow$$
$$\forall t : traces\ b.\ F\ \underline{sel}\ \alpha t \Leftrightarrow t\ (\underline{accepts}\ b)\ e\ \&$$
$$\alpha F = \alpha b$$

where

$$\alpha \in \mathrm{FILTER} \to \mathbb{P}(E)$$
$$\alpha F = \bigcup\{e \cup e' \mid (e, e') : F\}$$

The 'no junk' condition is useful when we come to construct filters for particular shapes of behaviour

Then, from the basic observations

$$e, e' \in E;\ e' \neq e$$
$$\vdash e \ (\underline{\text{has filter}}\ \text{SKIP})\ \Phi_{FPAIR}$$
$$\vdash e \ (\underline{\text{has filter}}\ e) \quad \{(\Phi_E, \{e\})\}$$
$$\vdash e' \ (\underline{\text{has filter}}\ e) \quad \Phi_{FPAIR}$$

it follows that

$$\mathrm{Reports}(r, p, v)(\underline{\text{has filter}}\ \mathrm{Reports}(r, p, v))$$
$$\{(\Phi_E, \{\mathrm{Reports}(r, p, v)\})\},$$

$$\mathrm{Reports}(r, p, v)$$
$$(\underline{\text{has filter}}\ \Box\{\mathrm{Reports}(r, p, v) \mid v : V\})$$
$$\{(\Phi_E, \{\mathrm{Reports}(r, p, v \mid v : V)\})\}$$

by applying lemma

$$H: \mathrm{Deterministic}(b_1, b_2);$$
$$\vdash e \ (\underline{\text{has filter}}\ (b_1 \,\Box\, b_2)$$
$$\{(s, s \cup \{e : E \mid e\ \underline{\text{starts}}\ b_2\}) \mid (s, s') : F_1\} \cup$$
$$\{(s, s \cup \{e : E \mid e\ \underline{\text{starts}}\ b_1\}) \mid (s, s') : F_2\},$$

$$\mathrm{Reports}(r, p, v)(\underline{\text{has filter}}\ \mathrm{REF})$$
$$\{\{\mathrm{Referee}(r, p)\}, \{\mathrm{Reports}(r, p, v \mid v : V\} \cup$$
$$\{\mathrm{Cancel}(r, p)\}\ )\}$$

by applying the previous lemma and

$$H;\ e\ \underline{\text{starts}}\ b_2;\ b_1\ \underline{\text{has ends}}\ F_1;\ \text{Independent}(b_1, b_2)$$
$$\vdash e\ (\underline{\text{has filter}}\ (b_1; b_2))$$
$$\{(s, s' \cup t') \mid (s, s'): F_1;\ t': \mathbb{P}(E)$$
$$\mid (\Phi_E, t') \in F_2 \} \cup$$
$$\{(s, s') \mid (s, s'): F_2 \mid s' \neq \Phi_E\}$$

and that this is also a filter in CRIS by applying

$$H;\ e \in \alpha\, b_1;\ \text{Independent}(b_1, b_2)$$
$$\vdash e\ (\underline{\text{has filter}}\ (b_1; b_2))\ F_1$$

$$H;\ e \in \alpha\, b_2;\ \neg e\ \underline{\text{starts}}\ b_2;\ \text{Independent}(b_1, b_2)$$
$$\vdash e\ (\underline{\text{has filter}}\ (b_1; b_2))\ F_2$$

$$H;\ e \in \alpha\, b_1;\ e \notin \alpha\, b_2;$$
$$\vdash e\ (\underline{\text{has filter}}\ (b_1 \| b_2))\ F_1$$

where

$$H \triangleq e\ (\underline{\text{has filter}}\ b_1)\ F_1;\ e\ (\underline{\text{has filter}}\ b_2)\ F_2$$

$$\text{Deterministic} \in \text{BEHAVIOUR} \leftrightarrow \text{BEHAVIOUR}$$
$$\text{Deterministic}(b_1, b_2) \Leftrightarrow$$
$$\forall e: E.\ e\ \underline{\text{starts}}\ b_1 \Rightarrow e \notin \alpha\, b_2\ \&$$
$$e\ \underline{\text{starts}}\ b_2 \Rightarrow e \notin \alpha\, b_1$$

$$\underline{\text{has ends}} \in \text{BEHAVIOUR} \leftrightarrow \text{FILTER}$$
$$b\ \underline{\text{has ends}}\ F \Leftrightarrow$$
$$\forall t: \text{traces}(b).\ F\ \underline{\text{sel}}\ \alpha t \Leftrightarrow \text{finals}\ b\ t$$

Determinism implies separability (see the previous section). behaviours of the form $e; b_1 \,\square\, b_1$ are also separable We can build a structural analysis for has ends by treating termination as some special event occurring and finding the filter for that event

This process, of going from simpler to more complex behaviours in stages, guessing at each step how to modify the filter to preserve its correctness, and then proving the construction correct, can be carried out for each event. We might expect to arrive reasonably quickly at a sufficient set of lemmas, covering all the special cases which have particularly compact forms of filter.

Finally, let us consider how to represent sets of events. We can gain more insight by looking at the results of the separate functions which collectively yield a result isomorphic to that of the original observation obs.

$$\begin{aligned} obs_S = {} & Submitted, Refereeing, Cancelled, Reports, \\ & Acknowledged, Decided, Informed, Scheduled, \\ & Suggested_A, Called, Intending, Suggested_D, \\ & Invited, Accepted, Replied \end{aligned}$$

What is the range of this function? If we look at one of the components, Reports for example, we can see that

$$Reports \in BTRACKS \rightarrow \mathbb{P}((P \times R) \times V)$$

which is conventionally written

$$Reports \in BTRACKS \rightarrow (P \times R \leftrightarrow V)$$

Such rewritings into relational form apply to all the observations involving cross products. In this case, we can go further. After some manipulation, we can write the range as

$$\begin{aligned} & ran\ Reports = \\ & \quad \{(rep\ Report)^{-1}(\alpha\ t) \mid \\ & \quad\ t : \{t \downarrow Report \underline{from}\ P \times R \times V \mid t : TRACES\}\} \end{aligned}$$

$$Reports \in BTRACKS \rightarrow (P \times R \nrightarrow V)$$

we need to demonstrate, for t in TRACES

$$G14 \triangleq \{Report(p, r, v_1), Report(p, r, v_2)\} \subset \alpha t \Rightarrow v_1 = v_2$$

We have by enumeration

$$t \in traces(?\,[]\,\{Report(r, p, v) \mid v : V\}) \vdash G14$$

and, since

$$\vdash CRIS \ \underline{sat} \ ?\,[]\,\{Report(r, p, v) \mid v : V\}$$

we can use the theorem concerning traces and satisfaction to give the desired result. Following this pattern in each case, we can show that

$$obs_S \in BTRACKS \rightarrow RANGE$$

where

$$\begin{aligned} RANGE = ((&P \nrightarrow A) \times (P \leftrightarrow R) \times (P \leftrightarrow R) \times (P \times R \nrightarrow V) \times \\ &\mathbb{P}(P) \times (P \rightarrow Dec) \times (P \nrightarrow A \times Dec) \times (T \rightarrowtail P) \times \\ &\mathbb{P}(A) \times \mathbb{P}(A) \times \mathbb{P}(A) \times \mathbb{P}(D) \times \\ &\mathbb{P}(D) \times \mathbb{P}(D) \times (D \nrightarrow Dec)) \end{aligned}$$

and further, that

$$\begin{aligned} &dom\ Refereeing \subset dom\ Submitted \\ &dom\ Reports \subset Refereeing \\ &Cancelled \subset Refereeing \\ &Disjoint(dom\ Reports, Cancelled) \\ &Acknowledged \subset dom\ Submitted \\ &\forall p : dom\ Informed; a : A; d : Dec \mid Informed\ p = (a, d). \\ &\quad Decided\ p = d\ \&\ Submitted\ p = a \end{aligned}$$

We can conclude that the set

$$DB_2 = RANGE \times E_{OOPS}$$

implements DB_1 in an obvious way

As a last step, we shall explore the sets which are isomorphic to the subset of RANGE defined by the theorems interrelating observations, to present the structure in a way more appropriate to a filing system and to attempt to have fewer unreachable states in the obvious implementation

In a manual filing system, we might consider allocating one drawer to files labelled by paper, containing the authorship, refereeing and reporting information relevant to that paper, another drawer to files on people (identified by elements of Q, which is henceforth to include A, D, and R) recording their status as delegates and authors, and a timetable to be filled in as scheduling progresses. Formally, we can define two sub-observations

$$obs_P = Submitted, Refereeing, Cancelled, Reports, Acknowledged, Decided, Informed$$

$$obs_Q = Suggested_A, Called, Intending, Suggested_D, Invited, Accepted, Replied$$

such that

$$obs_S = obs_P, Scheduled, obs_Q$$

Consider obs_P

$$obs_P \in BTRACKS \rightarrow RANGE_P$$
$$RANGE_P = ((P \rightarrow\!\!\!\!\!\!\bullet\ A) \times (P \leftrightarrow R) \times (P \leftrightarrow R) \times (P \times R \rightarrow\!\!\!\!\!\!\bullet\ V) \times \mathbb{P}(P) \times (P \rightarrow\!\!\!\!\!\!\bullet\ Dec) \times (P \rightarrow A \times Dec))$$

We want to convert elements of $RANGE_P$ to functions in

$$PF = P \twoheadrightarrow PAPER$$

for some (structured) set PAPER, which we can do in four reversible stages. The first is to convert each component of $RANGE_P$ to a partial function over P.

$$RP_1 = ((P \twoheadrightarrow Q) \times (P \twoheadrightarrow \mathbb{P}_1(Q)) \times (P \twoheadrightarrow \mathbb{P}_1(Q)) \times (P \twoheadrightarrow Q \twoheadrightarrow V) \times (P \twoheadrightarrow \{ACKD\}) \times (P \twoheadrightarrow Dec) \times (P \twoheadrightarrow (A \times Dec)))$$

$$stage_1 \in RANGE_P \rightarrowtail\!\!\!\!\!\!\rightarrow RP_1$$

$$stage_1 = Id \times \sharp \times \sharp \times \overset{CP}{} \times const\ ACKD \times Id \times Id$$

where the bijection property comes immediately from that of the components (of which $\sharp$, $\overset{CP}{}$ and const are defined in appendix A).

To turn this into a single function with cross-product target, the use of the combinator , is appropriate. We can show that this is bijective if it is restricted to functions with common domain, so the second stage is to reduce the functions to this form. It follows from the restrictions on RANGE that the domain of Submitted t includes that of any other function, for any t in BTRACKS. Domain extension can be done conveniently and reversibly by 'filling in the gaps' with a new value.

$$extend \in Y \rightarrow (\mathbb{P}(X) \times (X \twoheadrightarrow Y) \rightarrow (X \twoheadrightarrow Y))$$
$$extend\ y\ (S, f) = const(S, y) \oplus f$$

$$\vdash extend\ y \upharpoonright (S \times (S \twoheadrightarrow \overline{\{y\}})) \in S \times (S \twoheadrightarrow \overline{\{y\}}) \rightarrowtail\!\!\!\!\!\!\rightarrow (S \rightarrow Y)$$

$$RP_2 = ((P \rightarrow\!\!\!\!\!\!\rightarrow Q) \times (P \rightarrow\!\!\!\!\!\!\rightarrow \mathbb{P}(Q)) \times (P \rightarrow\!\!\!\!\!\!\rightarrow \mathbb{P}(Q)) \times (P \rightarrow\!\!\!\!\!\!\rightarrow Q \rightarrow V) \times (P \rightarrow\!\!\!\!\!\!\rightarrow \{ACKD\}_\perp) \times (P \rightarrow\!\!\!\!\!\!\rightarrow Dec_\perp) \times (P \rightarrow\!\!\!\!\!\!\rightarrow (A \times Dec)_\perp)$$

$$stage_2 \in RP_1 \rightarrowtail\!\!\!\!\!\!\rightarrow RP_2$$

$$stage_2(s, r, c, v, a, d, i) = (s, \text{extend}\,\Phi\,(\text{dom}\,s, r), \text{extend}\,\Phi\,(\text{dom}\,s, c), \text{extend}\,\Phi\,(\text{dom}\,s, v), \text{extend}\,\perp_{\{ACKD\}}(\text{dom}\,s, a), \text{extend}\,\perp_{Dec}(\text{dom}\,s, d), \text{extend}\,\perp_{A \times Dec}(\text{dom}\,s, i))$$

where, for all sets S

$$\perp_S \notin S$$
$$S_\perp = S \cup \{\perp_S\}$$

Then

$$RP_3 = P \rightarrow\!\!\!\!\!\!\rightarrow (A \times \mathbb{P}(Q) \times \mathbb{P}(Q) \times (Q \rightarrow\!\!\!\!\!\!\rightarrow V) \times \{ACKD\}_\perp \times Dec_\perp \times (A \times Dec)_\perp)$$

$$stage_3 \in RP_2 \rightarrow RP_3$$

$$stage_3(s, r, c, v, a, d, i) = s, r, c, v, a, d, i$$

Lastly, given

$$V_E = V \cup \{PENDING, CANCELLED\}$$

we can simplify the range of the resulting function

$$PAPER = (A \times (Q \rightarrow\!\!\!\!\!\!\rightarrow V_E) \times \{ACKD\}_\perp \times Dec_\perp \times \{VSENT\}_\perp)$$

$$stage_4 \in RP_3 \rightarrow PAPER$$

$$stage_4(s, r, c, v, a, d, \perp_{A \times Dec}) =$$
$$\quad (s, \text{extend PENDING } (r, \text{extend CANCELLED } (c, v)),$$
$$\quad a, d, (VSENT)_\perp)$$

$$i \in A \times D \Rightarrow stage_4(s, r, c, v, a, d, i) =$$
$$\quad (s, \text{extend PENDING } (r, \text{extend CANCELLED } (c, v)),$$
$$\quad a, d, VSENT)$$

and, since the composition of these stages over $ran\ obs_P$ is bijective, it is a relatively simple matter to determine how the events should update this new representation

Similarly, we can construct

$$QF \in Q \rightarrow PERSON$$
$$PERSON = (\{SUGGESTED_A\}_\perp \times \{CALLED\}_\perp \times \{INTENDING\}_\perp \times$$
$$\{SUGGESTED_D\}_\perp \times \{INVITED\}_\perp \times \{ACCEPTED\}_\perp \times$$
$$Dec_\perp$$

to give an implementation

$$DB_3 = PF \times QF \times (T \rightarrowtail\!\!\!\!\!\!\rightarrow P) \times E_{OOPS}$$

5 Summary

In the previous chapters, we have presented a scheme for describing the behaviour of the real world in terms of sequences of events communicated to a central point, and used it in the particular case of describing the process of conference organization. From this, we derived the sequences of events which could be fed into a machine equipped with a simple system for correcting errors in data entry, and described a set of observations which could be made from the machine and which would mimic the situation in the world. We then developed a design for the state of the machine, to a level from which the implementation in terms of arrays, files and other conventional data structures should be trivial.

The philosophy of our approach, of taking part of the world and describing its behaviour before embarking on a functional description of a system, is also that of the Jackson Systems Design method ([Jackson83]). This too has been strongly influenced by CSP, drawing on it as a structured approach to programming concurrent systems. A JSD specification is a collection of sequential programs with state, and the method produces programs which emphasise human comprehension rather than efficiency, although they are at least 'walkable' (to use Burstall's terminology). These specifications are then transformed systematically into programs more efficiently executable using current computer systems. In contrast, our approach draws on the more mathematical end of the CSP literature, and deals less systematically with implementation issues.

We can also compare the approach here with the constructive specification style of [Jones80]. The specification in that case would have started with a proposal for the state, perhaps that of DB_1, and operations corresponding to the events and defined on that state: that is to say, at a much later stage than here. Subsequent refinements would require the construction of a retrieve function to show which abstract state corresponded to a given representation. This then forms the basis for proofs of adequacy of the representation and correctness of the operations constructed on it.

The operations of the specification would include preconditions similar to those derived in the previous chapter to ensure their activation only at the correct time

The advantage of the method presented here is to make explicit information which is only a consequence of the operational definition The domain conditions of the operations are there to satisfy the requirement that operations should occur only in a given order. This requirement is stated in this specification and the consequences for the implementation derived later Certain incidental advantages follow OOPS was introduced by stating general properties, rather than as a separate operation on a specific state. We had at each level of implementation a characterization of the precise subset of the states which were actually reachable, which facilitated the refinement of the state by means of bijections, and hence guaranteed adequacy and made the definition of refined operations a more or less mechanical task We were also able to separate out the rather complex state oriented characterizations of domain from the rather simple actions of the operations.

The other point of comparison is CSP ([Hoare80, Hoare81]). The original incentive to develop the theory of the second chapter came from this work, although it was developed with the idea of describing a limited class of terminating behaviours, rather than potentially infinite processes There are thus differences in the interpretation of the terms: what we are doing is 'tapping' the line of communication between the committees and the rest of the world, and attempting to describe the possible sequences. The attempt may take the form of postulating various process-like entities, such as authors and referees, which may even make 'silent transitions' as a result of activity we cannot see.

There are some differences between the model suggested here and that of [Hoare80]. The use of '√' rather than explicitly separating out the final traces is, for finite behaviours, purely one of taste Our model is more explicit about alphabets, but allows the distinguishing of two behaviours by alphabet alone, which is hard to justify on the basis of experiments on

communications. Finally, the behaviours, unlike the processes of [Hoare80] as restricted by technical note (2), do not necessarily terminate deterministically. This allows us to use ?, which captures a common situation in a world where we are not privy to all the information which influences others.

The implementation is by no means complete, since we have deliberately abstracted from issues of the user interface. This was explicit in the choice of observations, and it is now time to honour our promise to consult the client. We have also made an implicit abstraction by saying that P and Q identify papers and people, without explaining how the operator is to communicate these identifiers to the system. Work remains to be done on formalizing these aspects of communication between user and system.

Appendix A
Some extensions to the theory of [Abrial81]

There are a few simple notions which we have used in the body of the specification which are not defined in [Abrial81]. They are presented here for completeness.

The first is the disjointness relationship between sets. The abstraction is rather more attractive in use than its definition.

$$\mathrm{Disjoint} \in \mathbb{P}(X) \leftrightarrow \mathbb{P}(X)$$
$$\mathrm{Disjoint}(S_1, S_2) \Leftrightarrow S_1 \cap S_2 = \Phi_X$$

The same applies to the second, which provides a definition of the alphabet of a sequence.

$$\alpha \in \mathrm{seq}(X) \rightarrow \mathbb{P}(X)$$
$$\alpha s = \mathrm{ran}\ s$$

Given an associative and commutative binary operator, and an element of its source set (usually a unit), there is a natural promotion to finite sets of operands.

$$\mathrm{Associative} \subset X \times X \rightarrow X$$
$$\mathrm{Associative}(*) \Leftrightarrow \forall x_1, x_2, x_3 : X \cdot x_1 * (x_2 * x_3) = (x_1 * x_2) * x_3$$

$$\mathrm{Commutative} \subset X \times X \rightarrow Y$$
$$\mathrm{Commutative}(*) \Leftrightarrow \forall x_1, x_2 : X \cdot x_1 * x_2 = x_2 * x_1$$

$$\text{continuation} \in (X \times X \to X) \times X \to (\mathbb{F}(X) \to X)$$
$$\text{Associative}(f) \;\&\; \text{Commutative}(f) \;\&\; x' \in S \Rightarrow$$
$$\quad \text{continuation}(f, x)(\Phi_X) = x \;\&$$
$$\quad \text{continuation}(f, x)\, S =$$
$$\qquad f(x', \text{continuation}(f, x)(S - \{x'\}))$$

Given an indexed set of homogeneous relations, we can convert a sequence of indices into a relation by composition.

$$\text{extension} \in (I \to (X \leftrightarrow X)) \to (\text{seq}(I) \to (X \leftrightarrow X))$$
$$\text{extension}\ F\ \langle\rangle = \text{Id}$$
$$\text{extension}\ F(i^\frown s) = F(i); (\text{extension}\ F\ s)$$

The remaining combinators are simple rearrangements of structures to put them into a more convenient form.

$$\text{const} \in \mathbb{P}(X) \times Y \gg\!\to (X \to Y)$$
$$\text{const}(S, y) = \lambda x : X \mid x \in S.\ y$$

$$\text{curry} \in (X \times Y \to Z) \gg\!\to (X \to (Y \to Z))$$
$$\forall (x, y) : \text{dom}\, f.\ \text{curry}\ f\ x\ y = f(x, y)$$

$$^{\text{CP}} \in (X \times Y \to\!\!\!\!\!\!\cdot\;\; Z) \gg\!\to (X \to (Y \to Z))$$
$$f^{\text{CP}} = (\text{curry}\ f) \downarrow \overline{\{\Phi\}}$$

$$^{\#} \in (X \leftrightarrow Y) \gg\!\to (X \to\!\!\!\!\!\!\cdot\;\; \mathbb{P}_1(Y))$$
$$\text{dom}\ \underline{r}^{\#} = \text{dom}\ \underline{r}$$
$$\forall x : \text{dom}\, \underline{r}.\ y \in \underline{r}^{\#}(x) \Leftrightarrow x\ \underline{r}\ y$$

References

[Abrial81] *A course on system specification*
J. R. Abrial (PRG internal document)

[Gustafsson81] *A declarative approach to conceptual information modelling*
M. R. Gustafsson, T. Karlsson, J. Bubenko
(SYSLAB Report 8)

[Guttag83] *An introduction to the Larch shared language*
J. V. Guttag (IFIP'83 Proceedings (to appear))

[Hoare80] *A model for communicating sequential processes*
C. A. R. Hoare (Technical monograph PRG-22)

[Hoare81] *A theory of communicating sequential processes*
C. A. R. Hoare, S. D. Brookes, A. W. Roscoe
(Technical monograph PRG-16)

[Jackson83] *System Development*
M. Jackson (Prentice-Hall International 1983)

[Jones80] *Software Development - A Rigorous Approach*
C. B. Jones (Prentice-Hall International 1980)

[Olle82] *Information systems design methodologies*
T. W. Olle, H. G. Sol, A. A. Verrijn-Stuart (*eds.*)
(North Holland 1982)

[Sufrin83] *Disjoint Unions, Recursive Types, Enumerated Types*
B. A. Sufrin (PRG internal document)

OXFORD UNIVERSITY COMPUTING LABORATORY
PROGRAMMING RESEARCH GROUP TECHNICAL MONOGRAPHS

SEPTEMBER 1983

This is a series of technical monographs on topics in the field of computation. Copies may be obtained from the Programming Research Group (Technical Monographs), 8-11 Keble Road, Oxford, OX1 3QD, England.

PRG-2 Dana Scott
Outline of a Mathematical Theory of Computation

PRG-3 Dana Scott
The Lattice of Flow Diagrams

PRG-5 Dana Scott
Data Types as Lattices

PRG-6 Dana Scott and Christopher Strachey
Toward a Mathematical Semantics for Computer Languages

PRG-7 Dana Scott
Continuous Lattices

PRG-8 Joseph Stoy and Christopher Strachey
OS6 – an Experimental Operating System for a Small Computer

PRG-9 Christopher Strachey and Joseph Stoy
The Text of OSPub

PRG-10 Christopher Strachey
The Varieties of Programming Language

PRG-11 Christopher Strachey and Christopher P. Wadsworth
Continuations, A Mathematical Semantics for Handling Full Jumps

PRG-12 Peter Mosses
The Mathematical Semantics of Algol 60

PRG-13 Robert Milne
The Formal Semantics of Computer Languages and their Implementations

PRG-14 Shan S. Kuo, Michael H. Linck and Sohrab Saadat
A Guide to Communicating Sequential Processes

PRG-15 Joseph Stoy
The Congruence of Two Programming Language Definitions

PRG-16 C. A. R. Hoare, S. D. Brookes and A. W. Roscoe
A Theory of Communicating Sequential Processes

PRG-17 Andrew P. Black
Report on the Programming Notation 3R

PRG-18 Elizabeth Fielding
The Specification of Abstract Mappings and their Implementation as B^+-trees

PRG-19 Dana Scott
Lectures on a Mathematical Theory of Computation

PRG-20 Zhou Chao Chen and C A R Hoare
Partial Correctness of Communicating Processes and Protocols

PRG-21 Bernard Sufrin
Formal Specification of a Display Editor

PRG-22 C A R. Hoare
A Model for Communicating Sequential Processes

PRG-23 C A R Hoare
A Calculus of Total Correctness for Communicating Processes

PRG-24 Bernard Sufrin
Reading Formal Specifications

PRG-25 C B Jones
Development Methods for Computer Programs including a Notion of Interference

PRG-26 Zhou Chao Chen
The Consistency of the Calculus of Total Correctness for Communicating Processes

PRG-27 C A R. Hoare
Programming is an Engineering Profession

PRG-28 John Hughes
Graph Reduction with Super-Combinators

PRG-29 C A R. Hoare
Specifications, Programs and Implementations

PRG-30 Alejandro Teruel
Case Studies in Specification: Four Games

PRG-31 Ian D Cottam
The Rigorous Development of a System Version Control Database

PRG-32 Peter Henderson Geraint A Jones and Simon B Jones
The Lispkit Manual

PRG-33 C A R. Hoare
Notes on Communicating Sequential Processes

PRG-34 Simon B Jones
Abstract Machine Support for Purely Functional Operating Systems

PRG-35 S D Brookes
A Non-deterministic Model for Communicating Sequential Processes

PRG-36 T. Clement
The Formal Specification of a Conference Organizing System