A Specification Formalism for Interacting Objects[*]

Olaf Owe
Department of Informatics, University of Oslo, Norway
email: olaf.owe@ifi.uio.no, fax: +47 22 852401

Abstract

We develop a logic for reasoning about requirement specification of objects with (internal and external) non-determinism. We try to achieve conceptual simplicity by avoiding the use of infinite sequences, restricting ourselves to finite traces, and by formulating a proof system satisfying the classical rules of first order logic. Under-specification is used to capture internal non-determinism; enabling us to state general facts about any execution. This reduces the language complexity: non-deterministic objects are specified by means of a simple event relation, called the ready relation, whereas the underlying semantics is a set of models, each with a set of such relations.

However, due to a non-standard interpretation, standard equational reasoning over object expressions is sound, and we obtain an expressive specification language, enabling us to express internal non-determinism, and enabling us to relate different execution points of several objects in the same specification formula, without the risk of making meaningless or inconsistent specifications. Our language is expressive enough to avoid the Brock-Ackerman anomaly and the merge anomaly.

We present a sound and (relatively) complete basic proof system. The classical rules and axioms of first order logic with equality are sound. In addition, some rules and axioms are needed for the ready relation and other object operators considered. Refinement may be done in two ways: By enriching a loose specification over a set of objects, one may refine several objects (reducing the set of possible models). By an explicit refinement operator, one may refine a single object (reducing the number of possible executions in each model).

1 Introduction

Our overall goal is to develop a practically useful formalism for specifying and reasoning about systems of concurrent objects. It is essential that the proof system be simple, and that the specification language be based on concepts that are intuitively clear and mathematically simple. We focus on requirement specifications, rather than abstract system design. The specification of a system should be expressed in terms of observable concepts; and the formalism should be strong enough to express liveness as well as safety properties, and be well suited for system development. The proof system should be compositional, and one should be able to

[*]Published in the Journal *Proceedings of the Estonian Academy of Sciences*, 4/3 (1998), pp. 198-215.

reason about objects from minimal requirements, for instance deriving properties of the concurrent composition of objects which are not yet fully specified.

We do want to present a general approach not depending on a specific programming language. We therefore consider an abstract notion of events. However, we are oriented towards object based systems, where objects have identity, which may be communicated; and we briefly introduce a notion of subclass. Conceptual simplicity of our specification language is achieved by using

- traces of observable events, rather than state variables,
- finite traces, rather than infinite ones,
- "execution dependent" *readiness relations*, allowing natural specification of liveness and of object dependencies,
- object equality, reflecting observable behaviour and "execution dependencies",
- generator induction, no fix-point operators.

The ready relations have a modal flavor, in the sense that their interpretation depends on a hidden quantified variable i, ranging over all possible "executions". This implies that specifications restrict a fixed but arbitrary execution. The ready relation denoted $\leftarrow$ expresses immediate readiness, whereas the "many-step" ready relation $\twoheadleftarrow$ expresses general liveness. By means of the object expression P/h, the continuation of object P after a finite trace h, we may express the readiness of P after h: $P/h \leftarrow x$ expresses that P is ready for x after accepting h in the "current" execution i.

We believe that avoiding infinite sequences at the specification level leads to more intuitive specifications, and it enables us to develop specifications that are close to (executable) abstract designs. For instance, a generator inductive definition of $P/h \leftarrow x$ gives an abstract design of P, characterizing the next possible actions x in the abstract state represented by h.

Under-specification is used to characterize non-deterministic object behavior; however, by means of *object equality* our language is expressive enough to state inherent non-determinism.

Simplicity of the proof system is achieved by using

- standard first order logic with equality and induction, but with a non-standard interpretation
- no temporal/ modal operators
- liveness reasoning by means of well founded induction
- no fix-point reasoning.

Even though the readiness relations have a modal flavor, they can not be nested; and we avoid the full complexity of modal operators. On the other hand, only certain kinds of fairness can be expressed.

Rather than using modal or temporal logic, we formulate a proof system consisting of the rules and axioms of *classical first order logic*, extended with standard rules for equality and induction, and with some additional proof rules and axioms axiomatizing object continuations and ready relations.

Under-specification of ready relations opens up for internal non-determinism. For instance, the specification $A \leftarrow a \oplus A \leftarrow b$ (where $\oplus$ denotes exclusive or) expresses that A is ready for either a or b in all executions, which allows for internal non-determinism. In contrast, $A \leftarrow a \wedge A \leftarrow b$ expresses that object A is ready for both event a and event b in all executions (expressing external non-determinism).

We believe this is intuitive and natural for programmers, since internal non-determinism is not explicitly observable, in the sense that from a single computation one cannot judge whether it is non-deterministic or not. This allows programmers to talk about deterministic and non-deterministic computations in the same way conceptually, and to state general facts about an arbitrary execution, which, we believe, leads to easily readable specifications.

We allow a specification formula to talk about the respective readiness of *several* objects, as in $P \leftarrow x \Rightarrow Q \leftarrow x$ (saying that in every execution Q is ready for anything P is ready for); in this respect our formalism is more liberal than **sat**-specifications of CSP [9, 2] and modal logics [13, 8, 12]. In order to be able to express such dependencies between objects, we use a semantics based on the notion of *execution families.*

By allowing object expressions and object equality, we may combine specifications as above with readiness characterizations of composed objects, as well laws similar to those in process algebra. As in CSP we let an object have an associated alphabet, and we consider objects with internal and external non-determinism. An object is said to be *deterministic* iff it is free from internal non-determinism. CSP operators such as $\|$, $\Box$, $\sqcap$ are used here with a semantics similar to that of CSP, but with object identity taken into account.

For instance by stating

$$(A \Box B) \leftarrow x \;\Leftrightarrow\; (A \leftarrow x \vee B \leftarrow x)$$

$$(\forall x \bullet (A \sqcap B) \leftarrow x \;\Leftrightarrow\; A \leftarrow x) \vee (\forall x \bullet (A \sqcap B) \leftarrow x \;\Leftrightarrow\; B \leftarrow x)$$

we express that in each execution the ready set of $A \Box B$ is the union of that of A and that of B; and that in each execution the ready set of $A \sqcap B$ is either that of A or that of B (and nothing else). Thus $\sqcap$ and $\Box$ correspond to internal and external choice, respectively. By means of object equality we may express laws such as

$$\begin{array}{lcl} (A \sqcap B)/h & = & (A/h) \sqcap (B/h) \\ (A \sqcap B) \Box C & = & (A \Box C) \sqcap (B \Box C) \\ (A \sqcap B) \| C & = & (A \| C) \sqcap (B \| C) \end{array}$$

stating that internal non-determinism can be expressed at the outermost level.

An appealing property of our proof system is that parallel composition corresponds to logical “and”, in the sense that

$$\forall h \bullet (P \| Q)/h \leftarrow \;\Leftrightarrow\; P/h \leftarrow \;\wedge\; Q/h \leftarrow$$

for objects P and Q with the same alphabet. This equivalence expresses that in an arbitrary execution the composition is accepting trace h if and only if both its components do. This close relationship between the composition and its components cannot be expressed with CSP's failure semantics.

Our framework is object-oriented in that objects have identity, which is reflected by object equality, and which may be communicated. For instance, if $P = A \| B$ and $Q = B \| C$, then P and Q are sharing object B, and the composition $P \| Q$ is the same as $A\|B\|C$, in contrast to process algebra. The composition $P\|Q$ has no other identity than those of its components. Thus we may talk about a sub-system, being a collection of objects without any other identity than those of its components, and may let a sub-system interface consist of several visible objects. This makes our formalism well suited for describing so-called system scenarios, specifying certain aspects of an object system, by describing the roles of the relevant objects, and their dependencies. In such scenarios the hidden (non-relevant) objects often cause dependencies between the visible objects; for instance in a multi-ticket-agent system discussed later, two agents may not offer the same seat at the same time.

Notation

We let $\langle x_1, x_2, .., x_n\rangle$ with $n > 0$ denote the sequence of x_1, x_2, .., and x_n. And ε denotes the empty sequence, whereas $\emptyset$ denotes the empty set. We use the following infix notations: ; denotes concatenation of two sequences, $\vdash$ denotes appending an element to the right of a sequence, $\upharpoonright$ denotes reduction of a sequence by a set (ignoring elements outside the set).

We will use x and y to denote events, s to denote sets of events, h to denote finite sequences of events, A and B to denote object expressions, and P and Q to denote object constants, from a set $\mathcal{P}$ of object names. For each object expression A there is an associated alphabet αA. We let p and q denote formulas; and $\vee$, $\wedge$, $\Rightarrow$ and $\neg$ denote logical or, and, implication and negation, respectively. We let $A \not\leftarrow$ denote $\neg A \leftarrow$.

2 Informal motivation and comparison to related work

In contrast to [3, 5, 7] based on stream processing functions, we avoid infinite sequences; liveness is expressed by a "many-step" ready relation ($\twoheadleftarrow$). Since we do not split the trace in several input and output streams, [1] it is then trivial to specify fair merge [4], since the total order of events is given in our traces. Furthermore, we avoid the Brock-Ackerman anomaly [1], since we are concerned with the total order of "input" as well as "output" events: We may distinguish the readiness of $P/\langle a, b, a'\rangle$ from that of $P/\langle a, a', b\rangle$ when a and b are input-events and a' is an output-event; for instance we may express that the former object continuation is ready for both output events a' and b', and the latter for a' only — which solves the Brock-Ackerman anomaly.

In contrast to other formalisms (such as [3, 7]) using under-specification to describe internal non-determinism, we may express whether an object is inherently non-deterministic or inherently deterministic (see section 4.1), due to the presence of object equality.

Our formalism satisfies the substitution property since the $\leftarrow$ relation is deterministic in each execution. We thereby avoid the difficulties of the general case of "non-deterministic functions/relations" discussed in [18]. In contrast to [3, 5, 7], we allow object equality; and as a consequence, object equality must satisfy

$$A = B \Rightarrow (\forall h \bullet A/h \leftarrow \Leftrightarrow B/h \leftarrow)$$

[1]In an abstract design where several in- or out-streams are desirable, these may be described as suitable projections of the object trace.

by standard equality reasoning. Thus equal objects must have the same ready set in each execution, which is true for identical objects. This equality relation is clearly stronger than observational equivalence. By stating

$$P/h1 = P/h2$$

we express that P behaves the same after $h1$ as after $h2$ in every execution. In this way a specification may relax the total ordering of events present in histories, to a partial ordering (imposed through a predicate q):

$$\forall h1, h2 \bullet q(h1, h2) \Rightarrow P/h1 = P/h2$$

Our ready relation operator may be used to express the immediate possibilities of an object in much the same way as the basic modal operators of Hennessy-Milner logic [8, 12]. And by means of the many-step ready relation ($\twoheadleftarrow$) we may express temporal behavior [17]. Thus like modal or temporal logic we may talk about the possibilities and the future of an object at a given execution point. But in contrast to such logics, we may, in the same formula, talk about the possibilities/future of several objects. We find this practically useful in specification. And we avoid the logical complexity of modal operators (at the cost of reduced expressive power).

In contrast to formalisms such as I/O-automata, state machines, UNITY, rewriting logic [14], and TLA [11], we avoid state variables, which makes our formalism more suitable for requirement specifications. Like rewriting logic, we may directly express action dependencies between objects. However, rewriting logic is not (yet) oriented towards specification purposes (for instance there is no simple way of expressing eventuality).

As shown in [16], failure semantics is not able to distinguish between an object which accepts a finite number, non-deterministically chosen, of a's and then accepts b and stops, and an object which non-deterministically chooses between a and b, stops after b, and repeats itself after a. We may distinguish the two, because the semantics of the former will be a set of executions each allowing b after finitely many (consecutive) a-steps, whereas the semantics of the latter includes an execution which never allows b. At the specification level, we may distinguish the two by means of the “many-step” ready relation ($\twoheadleftarrow$), since the specification $P \twoheadleftarrow b$ will imply that P must be ready for b in finite time.

In [16] it is also shown that failure semantics cannot express specifications such as

$$P/\langle a \rangle \leftarrow b \;\Leftrightarrow\; P \leftarrow b$$

stating that P is ready for b after a in an execution if, and only if, P was ready for b initially. Our formalism is well suited for expressing such relationships. Moreover, such specifications are useful, as later demonstrated by examples.

The refinement method inherent in loose specifications allows us to refine an object system. In addition we may refine individual objects by an explicit refinement operator, defined by subset of executions. In contrast to the subset relation on failure sets, our refinement relation will preserve (many-step) liveness results, as discussed below. This may be taken as another argument in favor of execution family semantics rather than failure semantics.

3 Formalization

We will give proof rules and axioms for sequents of form $l \vdash p$ with an "assumption" part l (a list of formulas) and an "assertion" part p (a single formula). (Lists in the assertion part could easily be allowed as well.) A *specification* S of a set of objects, $\mathcal{P}$, is a set of sequents with object constants from $\mathcal{P}$ only, and describes a first order theory with the sequents as non-logical axioms; the notation $l \vdash_S p$ means that $l \vdash p$ is provable in this theory.

Since we wish to be flexible with respect to underlying programming language, the semantics given below, based on execution families, is as abstract as possible, letting object equality of the specification language be interpreted by object equality of the model, and such that any (legal) model is obviously implementable.

3.1 Specification language

Our basic specification language is that of first order logic with equality, and with the above notation for event sequences, as well as the object expressions and object relations below. We consider first the following kinds of object expressions:

- P , an object constant (with given alphabet)
- A/h , the continuation of A after h (for object expression A), letting $\alpha(A/h)$ be αA
- 0_s , the "dead object" with alphabet s
- $\bot_s$, the "meaningless object" with alphabet s (same as $0_s/\langle a\rangle$ for $a \in s$).

Later we add parallel composition ($\|$). We consider the following relations over object expressions:

- $A \leftarrow$, expressing that object expression A is meaningful in "this execution"
- $A = A'$, expressing object equality

We abbreviate $A/(h \vdash x) \leftarrow$ to $A/h \leftarrow x$, and we introduce the notation $A \twoheadleftarrow x$, expressing that object A eventually will be ready to accept x. Since our language does not allow object variables, there is no quantification over objects. (However, object meta variables used in axioms and proof rules follow the usual substitution rules.)

For convenience we will overload all the above object operators, using the same symbols at the semantic level as well as at the specification level, except $\leftarrow$ which is available at the specification level only. As explained below the $\leftarrow$ relation is interpreted as a relation variable ranging over a family of relations $\underset{i}{\leftarrow}$, $i \in \mathcal{I}$, defined at the semantic level.

3.2 Validity

A model for a set of objects $\mathcal{P}$ is defined below as a non-empty family of executions $\mathcal{I}$, and will define the meaning of all object expressions over $\mathcal{P}$ and object relations, except for $\leftarrow$; instead it defines a family of relations $\underset{i}{\leftarrow}$, $i \in \mathcal{I}$.

Definition (Validity). A sequent $\vdash p$ from the specification language is *valid in a model* if

$$\forall i : \mathcal{I} \bullet p_i$$

is satisfied for all values of the free variables, where p_i is p with every occurrence of an $\leftarrow$ replaced by $\underset{i}{\leftarrow}$. And a sequent $\vdash p$ is said to be *valid*, denoted $\models p$, if it is valid in all models for $\mathcal{P}$. And we define $l \models p$ by $\models l \Rightarrow p$, taking commas in l as "and"s. Thus free variables in l or p have the whole sequent as scope, and are implicitly universally quantified.

Similarly, a *model for a given specification* over $\mathcal{P}$, is a model for $\mathcal{P}$ in which all non-logical axioms of the specification are valid. For a given specification S, a sequent $\vdash p$ is valid, denoted $\models_S p$, if it is valid in every model for S.

Notice that it may well be that neither p nor $\neg p$ is valid in a model, even when p is closed, since $(\forall i : \mathcal{I} \bullet p_i) \vee (\forall i : \mathcal{I} \bullet \neg p_i)$ need not hold.

3.3 Modeling

We will use a simple form of interleaving semantics: We imagine that an execution takes place event by event and that at each execution point each object is able to accept a subset of the set of events in its alphabet. Due to (internal and external) non-determinism many executions may be possible.

The semantics of a deterministic object can be described by its trace set, i.e. a prefix-closed set of finite traces. External non-determinism is reflected by the presence of several traces which are not prefixes of each other. For a non-deterministic object this is not sufficient, since a possible deadlock might be invisible in the trace set. However, in *each execution*, a non-deterministic object behaves as a deterministic object. Thus we may describe a non-deterministic object by the set of its possible "deterministic instances", each given by a prefix-closed trace set. As all non-determinism may be expressed at the outermost level, a non-deterministic object P may be regarded as $P_1 \sqcap P_2 \sqcap \cdots$ where each P_i is (internally) deterministic.

However, if the semantics of a set of non-deterministic objects is given by a set of trace-sets for each object, there is no way to express that a choice made by one object may depend on a choice made by another object. For instance, a scenario describing the user interface of two ticket agents may require that the agents may not both offer the same ticket at the same time. And a tank control system scenario may require that the "high-level detector" does not trigger at the same time as the "low-level detector".

In order to capture such dependencies between the objects, we describe a set of non-deterministic objects $\mathcal{P}$ by a set $\mathcal{I}$ of mappings, each associating a prefix-closed trace set to each object in $\mathcal{P}$, forming a possible combination of deterministic instances of the given objects. We will talk about $\mathcal{I}$ as the set of possible *executions*, since an element in $\mathcal{I}$ characterizes an (open) execution (where an environment may select between the possible traces).

Let $[P]_i^*$ denote the trace set of object P in execution i. The semantics of a set of objects is then given by

$$\lambda i : \mathcal{I}, P : \mathcal{P} \bullet [P]_i^*$$

We require that each $[P]_i^*$ is prefix closed, that no trace may contain events outside αP, and that I is non-empty. No other general requirements are needed.

The trace set of an object continuation is defined in the obvious way, letting

$$h' \in [P/h]_i^* \;\Leftrightarrow\; h; h' \in [P]_i^*$$

Notice that our modeling formalism allows us to distinguish between the trace sets $\{\varepsilon\}$ and $\emptyset$. We introduce a constant $\bot_s$ to denote the totally meaningless object with alphabet s, defining

$$[\bot_s]^*_i \triangleq \emptyset \qquad \text{DEF}(\bot)$$

and we let 0_s denote the totally dead (but meaningful) object, with alphabet s, defined by

$$[0_s]^*_i \triangleq \{\varepsilon\} \qquad \text{DEF}(0)$$

Two object expressions with the same alphabet are equal if they have the same meaning:

$$A = B \triangleq \forall i \bullet [A]^*_i = [B]^*_i \qquad \text{DEF}(=)$$

We may now express the total trace set of A by $\{h \mid A/h \neq \bot_{\alpha A}\}$, or equivalently by

$$Tr(A) \triangleq \{h \mid \exists i : \mathcal{I} \bullet h \in [A]^*_i\} \qquad \text{DEF}(Tr())$$

An object is said to refine another with the same alphabet if each possible trace set of the former also is possible for the latter

$$A \sqsubseteq B \triangleq \forall i\ \exists j \bullet [A]^*_i = [B]^*_j \qquad \text{DEF}(\sqsubseteq)$$

which is the same as $\{[A]^*_i \mid i \in \mathcal{I}\} \subseteq \{[B]^*_i \mid i \in \mathcal{I}\}$. Two object continuations with the same alphabet are *similar* if they have the same sets of trace sets:

$$A \approx B \triangleq A \sqsubseteq B \wedge B \sqsubseteq A \qquad \text{DEF}(\approx)$$

which is the same as $\{[A]^*_i \mid i \in \mathcal{I}\} = \{[B]^*_i \mid i \in \mathcal{I}\}$. This means that A has the same possible behaviours as B. A subclass relation, $A \leq B$, can be defined by requiring that $\alpha B \subseteq \alpha A$ and $\{(h \restriction \alpha B) \mid h \in [A]^*_i \wedge i \in \mathcal{I}\} \subseteq \{[B]^*_i \mid i \in \mathcal{I}\}$. Thus a subclass may extend the alphabet and add trace restrictions.

Notice that $P \sqsubseteq P \sqcap Q$, but not $P \Box Q \sqsubseteq P \sqcap Q$. Since refinement does not introduce any deadlocks, a refinement of A will preserve typical liveness properties, stating that in each execution A will be ready for something in finitely many steps, without deadlocking.

Readiness

We let the notation $A \underset{i}{\leftarrow}$ denote $\varepsilon \in [A]^*_i$, expressing that A is meaningful (or reachable); then $A/h \underset{i}{\leftarrow}$ is equivalent to $h \in [A]^*_i$, and $[A]^*_i$ is the same as $\{h \mid A/h \underset{i}{\leftarrow}\}$. Thus the unary $\underset{i}{\leftarrow}$ relation gives the same expressiveness as $[...]^*_i$, and all laws and definitions above may be restated in terms of the unary $\underset{i}{\leftarrow}$.

We abbreviate $A/\langle x\rangle \underset{i}{\leftarrow}$ to $A \underset{i}{\leftarrow} x$, expressing that A is immediately ready for event x in execution i. We may express that A is ready for event x in at most n steps in execution i:

$$A \overset{1}{\underset{i}{\twoheadleftarrow}} x \triangleq A \underset{i}{\leftarrow} x \qquad \text{DEF}(\overset{1}{\underset{i}{\twoheadleftarrow}})$$

$$A \overset{n+1}{\underset{i}{\twoheadleftarrow}} x \triangleq A \underset{i}{\leftarrow} x \vee [A]^1_i \neq \emptyset \wedge (\forall y \bullet A \underset{i}{\leftarrow} y \Rightarrow A/\langle y\rangle \overset{n}{\underset{i}{\twoheadleftarrow}} x) \quad \text{DEF}(\overset{n+1}{\underset{i}{\twoheadleftarrow}})$$

where $[A]^1_i$ denotes $\{x \mid \langle x\rangle \in [A]^*_i\}$. In order to capture unbounded readiness, we define $\underset{i}{\twoheadleftarrow}$ as the least fixpoint of the recursive equation

$$A \underset{i}{\twoheadleftarrow} x \;\triangleq\; A \underset{i}{\leftarrow} x \vee [A]^1_i \neq \emptyset \wedge (\forall y \bullet A \underset{i}{\leftarrow} y \Rightarrow A/\langle y\rangle \underset{i}{\twoheadleftarrow} x) \qquad \mathrm{DEF}(\underset{i}{\twoheadleftarrow})$$

with *false* as the minimum. Then $A \underset{i}{\twoheadleftarrow} x$ expresses that A eventually will be ready for x in execution i. We have $(\exists n \bullet A \overset{n}{\underset{i}{\twoheadleftarrow}} x) \Rightarrow A \underset{i}{\twoheadleftarrow} x$. (The converse need not hold when αA is infinite.)

For meaningful P, binary $\underset{i}{\leftarrow}$ has the same expressiveness as unary $\underset{i}{\leftarrow}$. Thus in the (normal) case that all object constants in $\mathcal{P}$ are meaningful, we may express all laws and definitions above in terms of binary $\underset{i}{\leftarrow}$.

4 Basic proof system

All classical rules of Gentzen style sequent calculus, with equality, are sound (including structural rules and axioms, elimination and introduction rules, as well as substitution rules and equality rules and axioms).[2] This can be proved by trivial elimination and introduction of the quantified i-index. However, Skolemization is not sound, unless the i-index is taken into consideration!

In addition, standard induction rules are sound, and we may reason about recursive definitions as usual. For recursive definitions not involving the ready relation, this is obvious. For recursive definitions of ready relations, all occurrences of a ready relation in a recursive definition must be interpreted by means of the same execution — which is reasonable.

We focus below on rules and axioms involving the ready relation, presenting first a basic system and then some extensions. Our proof system consists of all classical axioms and proof rules from first order sequent calculus (for instance as in [6]) and in addition the ones below.

In the proof rules below, we will use the convention that a rule of the form

$$\frac{\vdash p}{\vdash q}$$

abbreviates

$$\frac{a \vdash p}{a \vdash q}$$

where a is a list of formulas without occurrences of $\leftarrow$, i.e. assumptions without $\leftarrow$ are inherited. Some, but not all rules, allow unrestricted inheritance of assumptions; in such cases, the inheritance is shown explicitly in the rules. Notice that in this logic, even if $\vdash p$ follows from $\vdash q$, it may be unsound to conclude that $\vdash \neg q$ follows from $\vdash \neg p$ (due to the underlying quantifier).

Below, x, y, h, A, B, p and l are meta variables ranging over terms (i.e. x and y range over terms denoting events, h ranges over terms denoting histories, A and B range over object terms, p ranges over formulas, and l ranges over lists of (any) formulas).

The following rule follows from first order logic with equality:

[2]This also includes rules such as proof by cases, proof by contradiction, cut, resolution, etc.

$$\frac{l \models A = B \qquad l \models p(B)}{l \models p(A)}$$

where $p(A)$ denotes a predicate with any number of occurrences of A. (Here the explicit l signifies unrestricted inheritance of assumptions.)

4.1 Proof rules and axioms

The basic axioms about continuations and unary $\leftarrow$ are

$$\models A/\varepsilon = A \tag{A1}$$

$$\models A/h/h' = A/h;h' \tag{A2}$$

$$\models A/h \leftarrow \;\Rightarrow\; h \in (\alpha A)^* \tag{A3}$$

$$\models A/h \leftarrow \;\Rightarrow\; A \leftarrow \tag{A4}$$

$$\models \bot_s \not\leftarrow \tag{A5}$$

$$\models 0_s \leftarrow \tag{A6}$$

$$\models \forall x \bullet 0/\langle x \rangle \not\leftarrow \tag{A7}$$

$$\frac{\models \forall h \bullet A/h \leftarrow \;\Leftrightarrow\; B/h \leftarrow}{\models A = B} \text{ provided } \alpha A = \alpha B \tag{equality}$$

Thus, for two object expressions (with the same alphabet) we have $\models A = B$ if and only if $\models \forall h \bullet A/h \leftarrow \;\Leftrightarrow\; B/h \leftarrow$ (the "only if" part follows by first order equational reasoning). Furthermore, we may derive $\models A \neq B$ from $\models \neg \forall h \bullet A/h \leftarrow \;\Leftrightarrow\; B/h \leftarrow$ by equational reasoning. However, we may not derive $\models \neg(\forall h \bullet A/h \leftarrow \;\Leftrightarrow\; B/h \leftarrow)$ from $\models A \neq B$ (which would be unsound, due to the underlying quantifier).

Traces

If we let the notation $h \in Tr(A)$ abbreviate $A/h \neq \bot$, we may derive the following lemmas and rule:

$$\models h;h' \in Tr(A) \;\Leftrightarrow\; h' \in Tr(A/h) \tag{A8}$$

$$\models A/h \leftarrow \;\Rightarrow\; h \in Tr(A) \tag{A9}$$

$$\frac{\models A/h \not\leftarrow}{\models h \notin Tr(A)} \tag{trace}$$

Notice that the assertion $\models \forall h : Tr(A) \bullet A/h \leftarrow$ expresses that A is internally deterministic, and

$$\models \exists h : Tr(A) \bullet h \neq \varepsilon \wedge A/h \not\leftarrow$$

expresses that A is inherently non-deterministic. (If A satisfies neither, it may be refined to a deterministic object as well as to a non-deterministic one.)

Readiness

We introduce the axiom

$$\vdash \forall h, x \bullet A/h \leftarrow x \;\Leftrightarrow\; A/h \vdash x \leftarrow \qquad \text{(A10)}$$

It follows that $\vdash A = B$ is equivalent to $\vdash \forall h, x \bullet A/h \leftarrow x \;\Leftrightarrow\; B/h \leftarrow x$ provided both A and B are meaningful and have the same alphabet. Notice that (A4) is equivalent to $\vdash A/\langle x\rangle \leftarrow y \Rightarrow A \leftarrow x$.

The presented logic is sound and relatively complete, in the following sense: Assume that we have rules so that $l \vdash p$ if and only if $l \models p$ for l and p without object expressions. Then, with the addition of the above axioms and rules we have $l \vdash p$ if and only if $l \models p$ for l and p involving no other object-related operators than $\leftarrow$, $/$, and object equality and trace.

The soundness of the above axioms and rules follows by our interpretation and the semantical definitions. The completeness proof (except for object equality) is similar to that of [17] (which uses a more complicated semantic model).

In the natural case where all user named object constants P are meaningful ($P \neq \perp$), all occurrences of unary $\leftarrow$ may be removed and re-expressed by means of binary $\leftarrow$. And all occurrences of unary $\leftarrow$ in the rules and axioms above may be reformulated with the binary $\leftarrow$ and object equality.

Letting Alive(A) denote $\exists x \bullet A \leftarrow x$, the presence of immediate and absolute deadlock is expressed by $\vdash \neg\text{Alive}(A)$, absence of immediate deadlock is expressed by $\vdash \text{Alive}(A)$, and absence of eventual deadlock is expressed by

$$\vdash \forall h \bullet A/h \leftarrow \;\Rightarrow\; \text{Alive}(A/h)$$

4.2 Eventual readiness

Readiness in several steps can be axiomatized as follows:

$$\vdash \forall x \bullet A \overset{1}{\twoheadleftarrow} x \;\Leftrightarrow\; A \leftarrow x$$

$$\vdash \forall x \bullet A \overset{n+1}{\twoheadleftarrow} x \;\Leftrightarrow\; \text{Alive}(A) \wedge (\forall y \bullet A \leftarrow y \Rightarrow x = y \vee A/\langle y\rangle \overset{n}{\twoheadleftarrow} x)$$

Well-founded induction can be used to deduce eventual readiness:

$$\frac{\vdash \forall h \bullet A/h \leftarrow \wedge (\forall h' \bullet h' \leq h \Rightarrow A/h' \not\leftarrow x) \Rightarrow \text{Alive}(A/h) \wedge \forall y \bullet A/h \leftarrow y \Rightarrow f(h \vdash y) < f(h)}{A \leftarrow \;\; \vdash \;\; A \twoheadleftarrow x}$$

where $\leq$ denotes the sequence prefix relation, and f is a total function into the natural numbers.

(The many-step ready relations used here are interpreted by the corresponding indexed many-step ready relations.)

5 Parallel composition

As usual we let $\alpha(A\|B)$ be $\alpha A \cup \alpha B$. For convenience, we will use a $+$ operation augmenting the alphabet of an object: $A + s$ behaves like A except that it always accepts events from $s - \alpha A$; thus $\alpha(A + s)$ is $\alpha A \cup s$. The $+$ operator will be used to explain the composition of objects with different alphabets; in particular, we will have that $(A\|B) = (A + \alpha B)\|(B + \alpha A)$, where the components of the latter composition have the same alphabets, and that

$$(A\|B) \leftarrow x \;\Leftrightarrow\; (A + \alpha B) \leftarrow x \wedge (B + \alpha A) \leftarrow x$$

Modeling

The object $A\|B$ is not disjoint from A or B. This dependency can be reflected in the execution family semantics, as follows:

$$\begin{array}{lcl} [A\|B]^*_i & \triangleq & [A + \alpha B]^*_i \cap [B + \alpha A]^*_i \\ [A + s]^*_i & \triangleq & \{h \in (\alpha A \cup s)^* \mid (h \upharpoonright \alpha A) \in [A]^*_i\} \end{array}$$

Clearly the defined families are prefix closed and do not contain events outside the respective alphabets. Notice that for both object operators, its $[]^*_i$-semantics is defined in terms of the $[]^*_i$-semantics of the components. This means that an execution of a parallel composition is explained from the behavior of the components in the same execution. This form of dependency cannot be expressed within failure semantics.

Axioms and proof rules

In addition to associativity and symmetry of $\|$, the following laws may now be proved sound:

$$\begin{array}{lrcll} \vdash & A\|A & = & A & \\ \vdash & (A\|B) & = & (A + \alpha B)\|(B + \alpha A) & \\ \vdash & (A\|B)/h & = & (A/(h \upharpoonright \alpha A))\|(B/(h \upharpoonright \alpha B)) & \text{, for } h \in (\alpha A \cup \alpha B)^* \\ \vdash & (A + s)/h & = & (A/(h \upharpoonright \alpha A)) + s & \text{, for } h \in (\alpha A \cup s)^* \\ \vdash & A + \emptyset & = & A & \\ \vdash & A + s & = & A + (s - \alpha A) & \\ \vdash & (A + s) + s' & = & A + (s \cup s') & \end{array}$$

It remains to give laws for ready relations over object expressions with the introduced operators:

$$\begin{array}{lrcl} \vdash & (A + s) \leftarrow x & \Leftrightarrow & x \in (s - \alpha A) \vee A \leftarrow x \\ \vdash & (A\|B)/h \leftarrow x & \Leftrightarrow & (A + \alpha B)/h \leftarrow x \wedge (B + \alpha A)/h \leftarrow x \end{array}$$

Thus $(A\|B)/h \leftarrow x$ may be reduced to a formula of the basic system of section 3. In particular, for objects A and B with the same alphabet we have

$$\vdash \quad (A\|B)/h \leftarrow x \quad \Leftrightarrow \quad A/h \leftarrow x \wedge B/h \leftarrow x$$

We have that $A\|B = B$ iff (all objects of) A is contained in B; thus the $\|$-operator enables us to express object set inclusion.

Example: Assume event a is in $\alpha A - \alpha B$, and that b is in $\alpha B - \alpha A$, then by the laws above, $A\|B \leftarrow a \;\Leftrightarrow\; A \leftarrow a$ and $(A\|B)/\langle b\rangle \leftarrow a \Rightarrow A \leftarrow a$, thus we may prove

$$(A\|B)/\langle b\rangle \leftarrow a \Rightarrow A\|B \leftarrow a$$

expressing that the composition is ready for a after b only if the composition was ready for a initially. This is not expressible within failure semantics, as shown in [16].

6 A ticket agent example

Consider a ticket agent A offering tickets to his/her customers. The alphabet of A includes events of the form $A(v)$ where A denotes the object name, and v identifies the seat. Customers may accept or reject such events. For instance if A is willing to offer seats 1, 2, 3, and a customer is willing to accept seats 2, 3, 4, then either $A(2)$ or $A(3)$ will take place.

For an event x, the notation $x.\mathrm{VAL}$ extracts the seat identification; and the dot-notation is lifted to event sequences by point-wise extension.

In order to ensure that no seat can be sold twice by an agent A, we state the following safety requirement:

$$\vdash \forall h \bullet A/h \leftarrow \;\Rightarrow\; \mathit{nonrep}(h.\mathrm{VAL}) \qquad (\mathrm{S}_A)$$

where $\mathit{nonrep}(q)$ expresses that the sequence q is free from repetitions (i.e. no element occurs more than once in q). The following liveness requirement states that unsold seats remain available:

$$\vdash \forall x, y, h \bullet A/h \leftarrow x \wedge A/h \leftarrow y \wedge x.\mathrm{VAL} \neq y.\mathrm{VAL} \Rightarrow A/h{\vdash}x \leftarrow y \quad (\mathrm{L}_A)$$

In a system with only one agent, a most service minded agent could be characterized by

$$\vdash \forall x, h \bullet A/h \leftarrow x \;\Leftrightarrow\; A/h \leftarrow \wedge\, x \in \alpha A \wedge x.\mathrm{VAL} \notin h.\mathrm{VAL} \qquad (\mathrm{D}_A)$$

which implies requirements S_A and L_A above (and gives a deterministic object).

Two agents

Consider a system with two agents A and B with disjoint alphabets, $A(v)$-events and $B(v)$-events respectively. We may wish to verify that the parallel composition $A\|B$ satisfies (some of) the requirements above. It is then reasonable to ensure that one agent cannot offer its customers the same seat as the other agent is offering (at any time). This gives an additional safety requirement:

$$\vdash \forall x, y, h, q \bullet A/h \leftarrow x \wedge B/q \leftarrow y \Rightarrow x.\mathrm{VAL} \neq y.\mathrm{VAL} \qquad (\mathrm{S}_{A,B})$$

From S_A, S_B, and $\mathrm{S}_{A,B}$, we may derive
$\vdash \forall h, h' \bullet A/h \leftarrow \wedge\, B/h' \leftarrow \;\Rightarrow\; \mathit{nonrep}(h.\mathrm{VAL}; h'.\mathrm{VAL})$ which is the same as $\mathrm{S}_{A\|B}$. In fact, we may prove that $\mathrm{S}_{A\|B}$ is equivalent to S_A, S_B, and $\mathrm{S}_{A,B}$. And we may prove that $\mathrm{L}_{A\|B}$ is equivalent to L_A and L_B.

A service minded system with two agents can be characterized by

$$\vdash \forall h, v \bullet (A\|B)/h \leftarrow \wedge\, v \notin h.\mathrm{VAL} \Rightarrow ((A\|B)/h \leftarrow A(v) \vee (A\|B)/h \leftarrow B(v)) \quad (\mathrm{D}_{A,B})$$

(assuming v ranges over seats). This last requirement together with $\mathrm{S}_{A\|B}$ will imply $\mathrm{D}_{A\|B}$, provided the difference between event $A(v)$ and $B(v)$ is abstracted away.[3]

Assume $\mathrm{S}_{A\|B}$. The combination of L_A, L_B and $\mathrm{D}_{A,B}$ would mean that the available tickets are split between the agents initially, and that the agents may not interchange tickets later. $\mathrm{D}_{A,B}$ without L_A and L_B would open up for agent cooperation (such as ticket exchange), whereas L_A and L_B without $\mathrm{D}_{A,B}$ would allow the agents to be gradually supplied with tickets on need (say by a hidden agent coordinator).

[3]Such abstraction may be formalized by a construct A **by** m, where m is an abstraction function on events, such that $\alpha(A \textbf{ by } m) = \{m(x) \mid x \in \alpha A\}$.

Agent cooperation

Since the agents above have disjoint alphabets, we are unable to express in what way the past of one agent may influence the future of the other. We could either introduce an agent coordinator (listening to A- and B-events), or extend the alphabets of A and B, such that they become non-disjoint. For instance, we may synchronize A and B by adding a special kind of "transfer" event, allowing transfer of unsold tickets by A to B.

An alternative may be to use asynchronous interaction between A and B: Let us explore this a bit further, and let agents A and B have the same alphabets, letting A accept all B-events

$$\models \forall h, v \bullet A/h \leftarrow \ \Rightarrow A/h \leftarrow B(v) \qquad (\mathrm{L2}_A)$$

and similarly for agent B. Then $\mathrm{L2}_A$ and $\mathrm{L2}_B$ reflect an asynchronous interaction between A and B. However, any one of S_A, S_B, and $\mathrm{S}_{A,B}$, would now cause inconsistency. Consider

$$\models \forall h, v \bullet A/h \leftarrow A(v) \Rightarrow v \notin (h.\mathrm{VAL}) \qquad (\mathrm{S'}_A)$$

Then $\mathrm{S'}_A$ and $\mathrm{S'}_B$ will imply $\mathrm{S}_{A\|B}$. Notice that something like $\mathrm{S}_{A,B}$ is not needed for this. Cooperation now allows A and B to offer the same tickets. In order to prove $\mathrm{L}_{A\|B}$ it still suffices with L_A and L_B; and L_A is ensured by the following two requirements:

$$\models \forall v, w, h \bullet A/h \leftarrow A(v) \wedge A/h \leftarrow A(w) \wedge v \neq w \Rightarrow A/h \vdash A(v) \leftarrow A(w)$$

$$\models \forall v, w, h \bullet A/h \leftarrow A(w) \wedge v \neq w \Rightarrow A/h \vdash B(v) \leftarrow A(w)$$

7 Conclusion

We have developed a trace-based and observation-based specification formalism for objects with (internal and external) non-determinism, avoiding infinite sequences, and state variables. At the specification level, an object is seen as a simple event relation (ready relation), even though the underlying semantics (in each model) is a set of such relations. This is done by letting each specification formula state properties of a single, arbitrary execution. Such specifications are significantly simpler than for instance CSP specification (by **sat**), where an object is specified by a relation over traces and sets!

In contrast to other approaches in the same direction, our formalism enables us to talk about object equality, and to relate several execution points of several objects, in the same specification formula, without the risk of making meaningless or inconsistent specifications as in [5]. And it becomes possible to specify inherent non-determinism. Our formalism is expressive enough to avoid the Brock-Ackerman anomaly as well as the merge anomaly.

We have a sound and (relatively) complete basic system, formulated within the framework of first order logic. In particular, the classical rules and axioms of first order logic with equality are sound. Skolemization, however, is not sound. An essential advantage is that parallel composition corresponds to logical "and" (when adjusted for differences in alphabets).

We have illustrated our formalism with some basic object forming operators. In [17] some more operators are considered, and a lift example (with many lifts and floors) is specified and designed. Since we avoid infinite sequences we may use ordinary generator induction to define functions over traces. This enables us to develop specifications close to abstract designs. An extension to real-time is being investigated.

Acknowledgment

The author is indebted to Ole-Johan Dahl for encouragement constructive criticism, and valuable suggestions on several versions of this work, and to Neelam Soundarajan for suggesting readiness. Tony Hoare, He Jifeng, Willem-Paul de Roever and Ketil Stølen deserve thanks for valuable comments. This work has been supported in part by the Esprit Basic Research working group 6112, COMPASS and the Norwegian Research Council.

References

[1] J.D. Brock and W.B. Ackerman: "Scenarios: a model of non-determinate computation." In J. Diaz and I. Ramos, editors, *Formalisation of Programming Concepts,* Springer-Verlag LNCS 107, 1981.

[2] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe: "A Theory of Communicating Sequential Processes." *JACM* 31 (1984) 560-599.

[3] M. Broy: "A Theory of Nondeterminism, Parallelism, Communication and Concurrency." *Theoretical Computer Science* 45 (1986) 1-61.

[4] M. Broy: "Nondeterministic data flow programs: how to avoid the merge anomaly." *Science of Computer Programming* 10 (1988) 65-85.

[5] M. Broy et al: "The Design of Distributed Systems: An Introduction to FOCUS." Report SFB 342/2-2/92, Technische Univ. München, 1992.

[6] O.-J. Dahl: *Verifiable Programming.* The Hoare Series, Prentice Hall, 1992.

[7] P. Dybjer and H. Sander: "A Functional Programming Approach to the Specification and Verification of Concurrent Systems." *Formal Aspects of Computing* 1 (1989) 303-319.

[8] H. Hennessy and R. Milner: "Algebraic Laws for Nondeterminism and Concurrency." *JACM* 32(1) (1985) 137-161.

[9] C.A.R. Hoare: *Communicating Sequential Processes.* The Hoare Series, Prentice Hall, 1985.

[10] G. Kahn: "The Semantics of a Simple Language for Parallel Processing." Proceedings of *Information Processing Congress 74*, North-Holland, pp. 471-475, 1974.

[11] L. Lamport: "The temporal logic of actions." *ACM Transactions on Programming Languages and Systems,* 16(3):872-923, 1994.

[12] K.G. Larsen: "Modal Specifications." In Springer-Verlag LNCS 407, 1990.

[13] Z. Manna and A. Pnueli: "Adequate Proof Principles for Invariance and Liveness Properties of Concurrent Programs." *Science of Computer Programming,* 4(3) (1984) 221-337.

[14] J. Meseguer: "Conditional rewriting logic as a unified model of concurrency." *Theoretical Computer Science* 96 (1992) 73-155.

[15] R. Milner: *Communication and Concurrency.* The Hoare Series, Prentice Hall, 1989.

[16] E.-R. Olderog and C.A.R. Hoare: "Specification-Oriented Semantics of Communicating Processes." *Acta Informatica* 23 (1986) 9-66.

[17] O. Owe: "Specifying Concurrency by Generalized Ready Relations." Research Report 176, Dept. of informatics, Univ. of Oslo, 1993.

[18] M. Walicki: "Algebraic Specifications of Nondeterminism." Ph.D. thesis, University of Bergen, 1993.