Topology, computer science and the mathematics of convergence

A.W. Roscoe*

Abstract

At first sight, computer science and topology seem unlikely bedfellows. We show that they are linked by the idea of convergence: in computer science we are often interested in processes which converge, often in spaces which are, mathematically speaking, rather unconventional. And point-set topology can be thought of as the study of convergence in general spaces. The main example we use to illustrate this is the way topology can be used to prove properties of the fixed points used in definition of recursive programs. This is developed in the final section for the case of CSP. We show there that it is sometimes possible to use fixed point induction to prove properties of a process which is not defined as a recursion, by synthesizing a recursion of which it is the least fixed point.

1 Introduction

That topology and computer science should be closely linked is evidenced by the number, and indeed, the variety, of papers presented in this volume. But at first sight it is an unlikely combination: computer science is a practical discipline, while topology, in particular point-set topology, has long been regarded as one of the 'purest' parts of pure mathematics. In this paper we aim to show topologists how their ideas are being applied by computer scientists, and to show theoretical computer scientists how concepts long studied in the abstract by topologists can provide insight into their own problems.

---

*Oxford University Computing Laboratory, 11 Keble Road, Oxford OX1 3QD, UK.
The author gratefully acknowledges that the work reported in this paper was supported by grant number N00014-87-G-0242 from the ONR.

Most of mathematics is rooted in one way or another in the natural, real or complex numbers. Even those parts which do not study them or structures over them are devoted to the study of objects – for example groups, rings and fields – which generalise them. And, of course, topology is no exception. For point-set topology can be said to be an extension to general spaces of the study of concepts such as convergence, proximity, completeness, continuity and connectedness from $\mathbf{R}^n$.

Perhaps the most fundamental of these concepts is convergence, for it is in this that the basic idea of an open or closed set arises in $\mathbf{R}^n$: a set X is closed if any convergent sequence of points from X has its limit in X, and open if any sequence of points converging to a point in X must intersect X. Thus the other concepts can all be re-stated in terms of convergence: for example $X \subseteq \mathbf{R}^n$ is connected if and only if there are no nonempty Y and Z such that $X = Y \cup Z$ and no sequence of points in either Y or Z converges to a point in the other.

In theoretical computer science one often constructs mathematical models for computer-related objects: usually the semantic values of programs or the data that is stored in a computer. These often look very unlike the real numbers, since, by the nature of computers, they are usually rather discrete. Topology can be used to study how these values relate to each other. Sometimes, on the other hand, as in the study of image processing, signal processing or numerical analysis, we use topology, or discrete generalisations of it, to help us understand how discrete data stored inside a computer relates to the continuous real world. There is, for example, a large literature on *digital topology* where continuous topological concepts are reinterpreted and exploited in images consisting of a number of grid points in two, three or more dimensions (see Kong and Rosenfeld's paper in this volume, for example).

When modelling the behaviour of computer programs we are often interested in limits of one sort or another, usually related to characterising a program's general behaviour as the limit of approximations – usually finite, or finitely observable behaviours. This comes most to the fore when deciding on the values of recursively defined programs, which are (either explicitly or implicitly) defined to be the solutions to fixed point equations such as

$$P = F(P)$$

where the function (or program context) F represents the body of the recursion, which makes recursive calls by using its argument. The results used to guarantee the existence of these fixed points in the underlying mathematical spaces all require the topological concepts of completeness and convergence: we set up a sequence of values that ought to converge to the fixed point and use completeness to ensure that it exists. The success of point-set topology is that it allows us to make sense of, and reason about, these concepts in spaces which are very different from those found in traditional mathematics.

The most commonly used results are versions of Tarski's theorem: a monotone function f from a complete partial order X to itself has a least fixed point. Here, a complete partial order (cpo) is one with a least element $\perp$ and where every directed set Δ (Δ is defined to be directed if every finite subset has an upper bound in Δ) has a least upper bound $\bigsqcup \Delta$. The convergent sequence here is found by defining $f^{\alpha}(\perp)$ for all ordinals α:

$$\begin{array}{rcl} f^0(\perp) & = & \perp \\ f^{\alpha+1}(\perp) & = & f(f^{\alpha}(\perp)) \\ f^{\lambda}(\perp) & = & \bigsqcup_{\alpha\in\lambda} f^{\alpha}(\perp) \end{array}$$

This forms an increasing sequence which cannot, by Hartog's Theorem, go on increasing strictly for ever. Thus some $f^{\alpha}(\perp)$ is a fixed point (and it turns out to be the least). If one additionally asserts that f is *continuous* (preserves the limits, i.e. least upper bounds, of directed sets), then the least fixed point is always reached by $f^{\omega}(\perp)$.

Also, an increasing number of computer scientists are making use of the contraction mapping theorem over complete metric spaces to generate their fixed points – see De Bakker and Rutten's paper in this volume, for example. This theorem, of course, is also proved by setting up a sequence that converges to the (in this case, unique) fixed point. It is interesting to note that, unlike the case of cpos where there was $\perp$, there is no distinguished element in a complete metric space from which to begin converging to the fixed point. Anyone familiar with the proof of the contraction mapping theorem will know that it starts of with an arbitrary point x and establishes that the sequence $\langle x, f(x), f^2(x), \ldots\rangle$ converges to the fixed point. This last observation will be important in the next section.

In both cases the key to setting up the fixed point theorem was to get an appropriate idea of convergence (with related completeness). Of course this gives us a link with topology. We will find shortly that we can – in both the partial order and the metric space cases – use topology to analyse the fixed points.

The topology in metric spaces is obvious, but in partial orders less so. Nevertheless there are several different topologies which can be used usefully over cpos. The one which ties best in with the way that the process described above converges on the least fixed point – through the order from below – is known as the Scott topology (for a longer discussion of this topology and its properties, see [Gierz*et al*1980]). It defines a set U to be open if and only if it is upwards closed ($x \in U$ and $x < y$ implies $y \in U$) and such that, if Δ is directed with $\bigsqcup \Delta \in U$, then $U \cap \Delta \neq \emptyset$. A set is Scott-closed if and only if it is downwards-closed and contains the limits of all its directed subsets. It is easy to show that a function between cpos is continuous if and only if it is continuous with respect to the Scott topology. Like some other topologies used in computer science, the Scott topology is T_0 but not T_1, and indeed the order used to generate the topology can be recovered from it: $x \leq y$ if and only if every open set that contains x also contains y. This last property is not one shared by metrics, where many different metrics give give rise to the same topology, or by other order topologies that we will meet later.

The need to understand convergence in mathematically unconventional spaces has been at the root of most of the uses of point-set topology in domain theory and semantics. It has not only been used to study convergence within domains that had been created by other means, but also as a tool for deciding which might be useful spaces to look at and for creating domains themselves. There have been attempts, in particular Smyth's work ([Smyth1988] and his paper in this volume), to unify the metric space and order-theoretic approaches by the use of *quasi-metric* spaces (where the symmetry axiom of metric spaces is dropped) and their generalisation *quasi-uniform* spaces. A topologically-based analogue of completeness that has been much used recently is provided by *sober* spaces – see, for example, [Johnstone1982] and Lawson's paper in this volume. The *sobrification* of a T_0 space (see Mislove's paper in this volume) has a good claim to giving the natural definition of completion over partial-order based spaces.

Topology has frequently been united with category theory in the search for classes of domains over which to solve *domain equations*. A domain equation is an identity/isomorphism such as

$$D \cong \mathcal{P}(A \times D)$$

(Which means: D is isomorphic to the powerset of the Cartesian product of A and D.) Domains which, like this one, are defined to be isomorphic to some construction involving themselves, are often required in programming language semantics. Topological considerations apply at several different levels when setting up the theories which allow us to solve them.

- We have to remember that the domains we finally end up with will be used to define programming language semantics. Thus the category of spaces over which one solves equations should have appropriate topological structure to allow us to find fixed points and analyse them. The majority of approaches have used either some class of cpos or a representation of one. It has, however, also been shown, e.g. [America and Rutten1988] and see de Bakker and Rutten's paper in this volume, that it is possible to solve domain equations over classes of complete metric spaces.

- The domain constructors used in the equations, such as $\mathcal{P}$ (powerset or *powerdomain*) and $\rightarrow$ (function space) sometimes appear inconsistent with our desire to solve the equations, because of cardinality considerations (i.e. Cantor's theorem). Topological ideas, motivated by the idea of what is computable – as the limit of a series of finite observations, are used to pare down spaces like $\mathcal{P}(D)$ and $D \rightarrow E$ so that the equations do become soluble.

  Typically this means restricting ourselves to closed sets and continuous functions (both with respect to appropriate topologies). For example, the function space over cpo models is usually interpreted as the set of Scott-continuous functions, while the powerdomain of a metric domain is usually identified with the set of all closed subsets under the Hausdorff metric.

- Finally, it is necessary to solve the equations themselves. A wide variety of classes of domain have been proposed which allow this, but the method of solution is almost always based, explicitly or implicitly, on inverse limits.

In the next two sections we will show how the topologies used to characterise convergence to fixed points in particular domains. In particular we will examine ways of using these topologies to justify methods for proving properties of the fixed points. In the next section we provide a general survey of these methods, while in the final section we show how one of the techniques can be used in a novel way to prove properties of programs that were not defined explicitly as a fixed point; typically they will be parallel networks of simple CSP processes.

2 Proving properties of fixed points

In the introduction we identified convergence within semantic models as the driving force behind topology in theoretical computer science. To illustrate this we will show in this section how topology can be exploited to prove properties of fixed points. In this section we have deliberately kept details of semantic domains and the languages they model to a minimum, in an attempt to keep it accessible to the topological audience.

Our discussions will be based around the following proof rule — whose validity or otherwise depends on the choice of healthiness conditions applied to body $F(\cdot)$ of the recursion and to the predicate $R(\cdot)$ we are trying to establish.

> Fixed point induction: Suppose that the property $R(\cdot)$ is satisfiable (i.e. not uniformly false) and satisfies healthiness condition H_R, and that $F(\cdot)$ satisfies healthiness condition H_F. Then, if
>
> $$\forall P.R(P) \Rightarrow R(F(P))$$
>
> (i.e. we can prove R holds of the body of the recursion on the assumption that it holds of all recursive calls), then infer $R(\mu P.F(P))$ (the value of the recursion – namely the appropriate fixed point of the function F).

(This principle is also known as *recursion induction.*)

Here, the assumption that the predicate is satisfiable corresponds to the usual inductive base case: without it, we could prove *false* true of anything. Intuitively, the condition H_F should be strong enough (under the assumption of satisfiability) to prove that there is some sequence of points, all satisfying R, converging to $\mu P.F(P)$ in some sense; and H_R should ensure that this is enough to establish $R(\mu P.F(P))$. H_R will thus usually state that $\{P \mid R(P)\}$ is closed in the topology determined by the mode of convergence used by H_F. By studying different types of space and different modes of convergence we will be able to establish a variety of pairs (H_F, H_R) that suffice.

The best-known form of this rule is based on the Scott topology which we met in the introduction (and known as Scott induction), and naturally applies to functions F from a cpo X to itself. All we have to assume of the function F is that it is monotone (i.e. just what is required to ensure a fixed point). A strong condition is required of R though: $R(\perp)$ must hold (reinforcing the satisfiability constraint) and, if Δ is a directed set all of whose members satisfy R, then $R(\bigsqcup\Delta)$ holds[1]. The assumption $R(P) \Rightarrow R(F(P))$ in the rule, plus the assumptions about R, combine to prove inductively that all of the $F^\alpha(\bot)$ that converge to, and include, the fixed point satisfy R.

The unfortunate feature of this form of induction is that the property must hold at $\bot$ — frequently the worst-behaved element of the models computer scientists build. One way of getting around this is to use not one predicate, but a series of strengthening ones. For a continuous F one would try to construct inclusive predicates R_n such that $R_0(\bot)$ holds, and $R_{n+1}(P) \Rightarrow R_n(P)$ and $R_n(P) \Rightarrow R_n(F(P))$ both hold for all n and P. It is then straightforward to prove that $R_n(F^m(\bot))$ holds whenever $m \geq n$, so that $\forall n.R_n(\mu P.F(P))$ holds.

Scott induction is based very clearly on the mode of convergence used in the proof of Tarski's Theorem. There is another version which is based in the same way on the contraction mapping theorem, and was introduced by the author in [Roscoe1982, Brookes*et al*1984].

If we can view the space X as a complete metric space, the body F of our recursion as contraction mapping, and $\{P \mid R(P)\}$ a closed subset, then the rule is valid, as is very simple to demonstrate. We know, by the contraction mapping theorem, that F has a unique fixed point in the space of all processes; but, as F preserves R, it is also a contraction mapping on the complete metric space $\{P \mid R(P)\}$ (a closed nonempty subset of a complete metric space is itself a complete metric space). As F has *a* fixed point in the subspace, the *unique* fixed point in the whole space must lie in the subspace.

An attractive feature of this induction is that the base case is totally arbitrary: all we need know is that there is *some* object satisfying R. (This is related to the observation we made earlier about convergence in a complete metric space being able to start from anywhere.) Even in models which are usually viewed as cpos, such as the untimed models of CSP [Brookes and Roscoe1985, Brookes*et al*1984], it is frequently useful to introduce a metric structure specifically to apply this version of recursion induction. In the case of the CSP models this metric is usually based on the number of communications which it takes to tell processes apart:

$$d(P,Q) = \inf\{2^{-n} \mid P{\downarrow}n = Q{\downarrow}n\}$$

where $P{\downarrow}n$ captures P's behaviour for its first n steps. The condition that

[1] Any R such that $\{P \mid R(P)\}$ is nonempty and Scott-closed satisfies both these constraints. A predicate R satisfying this directed limit condition is called called *inclusive* or sometimes *continuous*.

a recursion represents a contraction map then corresponds to it being *constructive*: the $n+1$-step behaviour of $F(P)$ must depend only on the n-step behaviour of P. Almost all reasonable, well-formed recursions which CSP 'programmers' devise turn out to be constructive. Chiefly because of the hiding operator $P\backslash X$ which makes all occurrences of events in X internal and so shortens traces, there are, however, many recursions one can write down which are not constructive. Thus the metric fixed-point theory is not, in this case, as general as the cpo one[2].

Notice that if we can prove a given partial-order based recursion contractive with respect to a metric, then we have proved that the least fixed point is actually the only (unique) fixed point. This can often be a very useful principle in itself. If P is any fixed point of the function F, then the property $R(Q) \equiv (Q = P)$ clearly inducts through F (i.e. $\forall Q.R(Q) \Rightarrow R(F(Q))$). Since only one of these predicates is actually true of the least fixed point, the inductive principle cannot be valid for proving equality with a single point unless F has a unique fixed point[3]. Thus the property of having a unique fixed point is actually closely tied in with the validity of our induction.

These metrics based on restriction operators (usually indexed by the natural numbers but sometimes on larger subsets of the real numbers[4], where restriction operators with higher index are strictly more discriminating and all distinct objects are distinguished by one of them, are very common in computer science. They have a number of special properties, the most notable of which is that they are *ultra-metrics* (sometimes called *non-Archimedian* metrics) which satisfy the following stronger version of the triangle inequality:

$$d(x,z) \leq \max(d(x,y), d(y,z))$$

Among the interesting properties of ultra-metrics is that all open balls $B(x,r) = \{y \mid d(x,y) < r\}$ are closed, and no two of these balls (with the same or different radii) intersect without one being contained in the other. We will return to the theory of these metrics later, briefly. It turns out that the closed sets (i.e. the ones admissible for recursion induction)

[2]Notice that, by looking at the same space both as a cpo and as a complete metric space, it has proved possible to deduce things from each view that were not obvious from the other.

[3]And notice that if the condition H_R on predicates is the same as $\{P \mid R(P)\}$ being closed in some T_1 topology then equality with a single point must be admissible.

[4]Occasionally restriction operators are used indexed by more exotic orders such as the countable ordinals, but these do not give rise naturally, if at all, to metrics.

are, in the case of a metric based on restriction operators, just the ones whose failure (for a given object) is detectable by one of the restrictions, or formally

$$\neg R(x) \Rightarrow \exists n.\forall y.(x{\downarrow}n = y{\downarrow}n \Rightarrow \neg R(y))$$

Fortunately it usually turns out that the predicates thus admissible include most of the ones one seeks to prove of processes in practice.

Two further examples where metrics based on restrictions have been used with this type of recursion induction are timed CSP [Reed1988, Reed and Roscoe1986, Reed and Roscoe1988] (where there is a restriction operator for every non-negative real) and finitely branching synchronisation trees [Brookes*et al*1986], frequently used to model the operational semantics of languages like CSP and CCS. These two examples turn out to be well-suited for similar reasons. In real-time CSP it turns out that every operator is non-expanding and every recursion is a contraction because no operator can see into the future, which would be necessary for it to push a pair of processes further apart.

Where, as in timed CSP, *every* recursion is a contraction map, the only conditions for a recursion induction to be valid are that the predicate represents a non-empty and closed set in the metric topology. In real-time CSP a predicate represents a closed set if and only if its failure (on any given process) is detectable by some finite time (which can vary from process to process). In other words, there must be some time t such that no process with the same t-restriction as the given one can satisfy the predicate. Thus any predicate which states that all of a process' finitely observable behaviours must satisfy some specification represents a closed set, and so is admissible for induction if it is satisfiable. Schneider [Schneider1990] has shown how in models like those of timed CSP where processes are identified with sets of behaviours one can often, when using the metric-space version of recursion induction, eliminate the need for proving satisfiability these *behavioural predicates* among the space of processes by expanding the models to contain non-processes, including one with no behaviours, thereby making satisfiability trivial. (The price one pays for this generalisation is that one cannot assume that P is a process when proving $\forall P.R(P) \Rightarrow R(F(P))$, but this rarely proves much of a handicap in practice.)

Over synchronisation trees, natural operators usually have the similar property that they never forget actions (though they may make them internal); the n-step behaviour of the input determines the n-step behaviour if the output, so to be constructive it is enough that some stage of the definition of a function adds a step of behaviour. Interestingly, the major use of the induction rule in [Brookes*et al*1986] was one over spaces of operators (functions from trees or tuples of trees to trees), where the operators were defined by a form of recursion, to show that the operators being defined were themselves nonexpanding or contractions. For the properties of being a constructive or non-destructive operator are admissible since any failure would show up in a finite number of steps. (There would be two inputs with identical n-step behaviours, the results of which, when one applied the offending operator, would have different $(n + 1)$- or n-step restrictions as appropriate.)

Before leaving the subject of the version of the rule based on ultra-metric spaces, it is worth pointing out another topological idea which can be of use in its application. This is that of *zero-dimensionality*. A T_1 space is said to have weak inductive dimension zero if, whenever a point x belongs to open U, we can find a clopen (closed and open) C such that $x \in C \subseteq U$. Every ultra-metric space trivially has weak inductive[5] dimension 0 because, as we have already remarked, all open balls $B(x, r) = \{y \mid d(x, y) < r\}$ are also closed. A space is said to have *strong* inductive dimension 0 if it satisfies the stronger property that any pair E, F of disjoint closed sets can be separated by a clopen set: i.e. there is a clopen set C such that $E \subseteq C$ and $F \cap C = \emptyset$. Notice how these definitions mimic the definitions of regular and normal spaces. Even though every metric space is normal, there is a famous example due to Roy [Roy1969] which shows that, in general, a metric space can have weak induction zero without having strong inductive dimension zero. However, it is easy to show that every ultra-metric space does have the stronger property. Given E and F as above, we can set $C = \bigcup\{B(x, r) \mid x \in E \wedge B(x, r) \cap F = \emptyset\}$. This set (which is obviously open) can be shown to be closed using the fact that if $y \in B(x, r)$ then $B(x, r) = B(y, r)$.

Why is this interesting when it comes to applying our rule? It is because of the condition which says that any allowable predicate must be satisfiable, which means that the set of all things satisfying it is nonempty. For we sometimes have to deal with predicates of the form $R_1 \wedge R_2$, where both R_1 and R_2 are satisfiable and represent closed sets, but where it is not clear that there is any object satisfying them both. What the above result says is that if there is none then there is a predicate S such that both S and $\neg S$

[5] We do not need to consider spaces with dimension greater than zero, and so do not define them. However, as is suggested by the names, the definitions we gave here of weak and strong inductive dimension zero are just the base cases of inductions for defining what it means for a space to have weak or strong inductive dimension n.

are admissible and where $R_1 \Rightarrow S$ and $R_2 \Rightarrow \neg S$, which can be useful[6].

We have already discovered that, on the assumption that equality with a given process is to be an acceptable predicate R, the 'well-definedness' condition H_F on recursions must at least imply uniqueness of fixed points. Interestingly, over complete lattices, it turns out to be possible to establish a version of fixed point induction that uses no stronger assumption. Over any partial order we can define the *interval topology* to be the smallest topology in which all the 'closed intervals' $[x, y] = \{z \mid x \leq z \leq y\}$ are closed sets. Now the interval topology over a complete lattice is compact (see [Gierz*et al*1980], for example), so that the following theorem can always be applied. (In its original reference cited below, this theorem was proved for a specific complete lattice, namely the traces model for CSP.)

THEOREM 2.1 ([ROSCOE1982]) *Suppose $\langle X, \leq \rangle$ is a complete lattice and that $\mathcal{T}$ is a compact topology on X which extends the interval topology. Suppose also that the monotonic function $f : X \to X$ has a unique fixed point x. Then if $Y \subseteq X$ is nonempty, closed in $\mathcal{T}$ and such that $f(Y) \subseteq Y$, we can deduce $x \in Y$.*

PROOF We will prove by transfinite induction that the sets $[f^\alpha(\bot), f^\alpha(\top)] \cap Y$ (which are closed) are all non-empty, where $\bot$ and $\top$ are respectively the least and greatest elements of X. ($f^\alpha(\bot)$ is defined as earlier, and $f^\alpha(\top)$ is defined by a symmetric iterative process starting from $\top$.)

- When $\alpha = 0$, the set equals Y, which is nonempty by assumption.
- If $x \in [f^\alpha(\bot), f^\alpha(\top)] \cap Y$ then

  $$f^{\alpha+1}(\bot) \leq f(x) \leq f^{\alpha+1}(\top)$$

  since $x \in [f^\alpha(\bot), f^\alpha(\top)]$ and by monotonicity of f. And $f(x) \in Y$ since $f(Y) \subseteq Y$. Hence

  $$[f^{\alpha+1}(\bot), f^{\alpha+1}(\top)] \cap Y$$

  is nonempty, for it contains $f(x)$.

[6] Of course, if both R_1 and R_2 individually induct through the body of the recursion, i.e. $\forall P.R_i(P) \Rightarrow R_i(F(P))$ for $i \in \{1, 2\}$, then it is not necessary to go through this analysis, since we then know that $R_1(\mu p.F(p)) \wedge R_2(\mu p.F(p))$ holds by two separate applications of the rule. Also, we have already discussed Schneider's technique for removing the satisfiability constraint from *behavioural* predicates.

- Finally, if each $[f^\alpha(\bot), f^\alpha(\top)] \cap Y$ is nonempty for $\alpha < \lambda$ (a limit ordinal) then, since these sets form a descending chain of closed sets in a compact space, their intersection is nonempty. But clearly

$$\bigcap_{\alpha<\lambda} ([f^\alpha(\bot), f^\alpha(\top)] \cap Y) = (\bigcap_{\alpha<\lambda} [f^\alpha(\bot), f^\alpha(\top)]) \cap Y$$

and it is not hard to see that

$$\bigcap_{\alpha<\lambda} [f^\alpha(\bot), f^\alpha(\top)] = [\bigsqcup_{\alpha<\lambda} f^\alpha(\bot), \sqcap \alpha < \lambda f^\alpha(\top)] = [f^\lambda(\bot), f^\lambda(\top)]$$

Hence $[f^\lambda(\bot), f^\lambda(\top)] \cap Y$ is nonempty, completing the induction.

Now from the proof of Tarski's theorem in the introduction we know that there is some α such that $f^\alpha(\bot)$ is the least fixed point. And by symmetry there is a β such that $f^\beta(\top)$ is the greatest fixed point. But, since we have assumed that f has a *unique* fixed point, we know $f^\alpha(\bot) = f^\beta(\top)$. If γ is the greater of α and β we know that $[f^\gamma(\bot), f^\gamma(\top)]$ contains only the fixed point. Since its intersection with Y is nonempty, the fixed point is contained in Y. ▮This proof depends crucially on the existence of a top element. Indeed the theorem is not true of more general partial orders. One need only consider the flat truth value domain $\{true, false, \bot\}$ and the continuous function $\neg$. This has unique fixed point $\bot$ and maps the closed set $\{true, false\}$ into itself.

This is unfortunate, since a great many of the partial orders one meets in computer science fail to have top elements. For a large class of these orders, moving up through the order corresponds to becoming more and more defined in some way, either losing nondeterministic choices or places where a program might loop. They usually contain many maximal elements, which are the values of fully defined programs. For example, the standard failures/divergences model of CSP can [Roscoe1988a] be ordered in two ways, one of which places all divergence-free processes (ones that can never loop) on top[7], and the other in which only the fully deterministic processes are maximal. In orders like these one can often deduce by some means that the least fixed point of a function is maximal (e.g. because it is divergence-free). Notice that if the least fixed point of f is maximal, then it is necessarily the

[7] A similar 'alternative order', for Dijkstra's language of guarded commands, has been developed independently in [Nelson1987]. It seems quite possible that strong orders of this type can be developed for other application as well.

only fixed point of f. There is a variant of Theorem 0.2.1 which allows us to prove properties of this type of fixed point.

The *lower* topology on any cpo X is the smallest one in which all sets of the form $x\uparrow = \{y \mid y \geq x\}$ are closed. (Notice that, on an order with a $\top$, the interval topology refines this one.) It is easy to show that it is compact for every consistently complete X[8].

THEOREM 2.2 ([ROSCOE1988A]) *Let X be a complete partial order and let $\mathcal{T}$ be a compact extension of its lower topology. Then, if $f : X \to X$ is a monotonic function whose least fixed point x is maximal (and hence unique) and Y, closed in $\mathcal{T}$, is such that $f(Y) \subseteq Y$, we have $x \in Y$.*

PROOF This is very similar to that of Theorem 0.2.1. Transfinite induction establishes that the (closed) sets $(f^\alpha(\perp))\uparrow \cap Y$ are all non-empty, where $\perp$ is the least element of X. But the fact that the least fixed point of f is maximal implies that $(f^\alpha(\perp))\uparrow$ eventually (i.e. for large enough α) contains only the fixed point. The result follows immediately. ∎It is mathematically pleasing to have characterisations of when the recursion induction rule is valid that are independent of a specific metric. Even though the great majority of recursions included in CSP programs are constructive, it proves useful to have these abstract forms of the rule when the function is derived from the process and not the other way round: this is discussed in the next section.

3 Induction over state-spaces

In this section we concentrate on CSP, a notation which originally motivated much of the work in the last section. The fact that the semantics of CSP has traditionally been given in abstract mathematical models with well-defined notions of convergence has frequently led to its being studied via topology. A number of references which relate directly to the subject of this paper have already been given. Two rather different applications of topology to CSP may be found in the papers by Barrett and Goldsmith, and by Mislove in this volume.

The application of the rules in the last section to processes defined explicitly by recursion has been well-documented in the literature. What we seek

[8]A cpo is consistently complete if every set with an upper bound has a least upper bound.

to do here is to show how they can sometimes be used to prove properties of processes which have not been defined by recursion, typically networks of processes communicating with their internal communications hidden.

This section assumes familiarity with the notation of CSP and its semantics in the failures/divergence semantics. A knowledge of operational semantics is also a prerequisite.

A simple example which we will use to illustrate and develop our techniques is provided by the following. If $B_1 = in?x \to out!x \to B_1$, consider the process

$$B_M = B_1 \gg B_1 \gg \cdots \gg B_1$$

where there are M copies of B_1. We will assume that $M > 1$. (The piping, or chaining operator $P \gg Q$ operates on two processes with alphabet $in.T \cup out.T$, and has the effect of putting P and Q in parallel with the out communications of P synchronised with the in communications of Q, and hidden. The in communications of P and the out communications of Q become the visible communications of the combination. The operator is associative.)

It should be obvious that B_1 is a one-place buffer and that B_M is an M-place buffer, in the sense that a k-place buffer never refuses an input if it contains less than k items, and never refuses to output (the oldest input it has not yet output) when it contains any items. For, an input to the leftmost B_1 will filter across to the rightmost via hidden communications irrespective of what else the system does or does not do. There are two types of result one might want to prove about the network B_M. Firstly, one might wish to prove it met some specification defined abstractly over a semantic model — we will be using the failures/divergences model for this example — and secondly one might want to prove that it either equalled or refined some other process.

A typical specification one might want to prove would be that it was a buffer, where a buffer (of type T) is defined to be a process whose failures and divergences $\langle F, D\rangle$ satisfy (i) $D = \emptyset$ (i.e. the process is divergence-free) and (ii) for all $(s, X) \in F$

$$\begin{aligned} s{\downarrow}out &\leq s{\downarrow}in && (1)\\ s{\downarrow}out = s{\downarrow}in &\Rightarrow X \cap in.T = \emptyset && (2)\\ (s{\downarrow}out)\langle a\rangle \leq s{\downarrow}in &\Rightarrow out.a \notin X && (3) \end{aligned}$$

Here, $s{\downarrow}in$ and $s{\downarrow}out$ are respectively the sequences of values in T communicated over the channels in and out within s. Condition 1 says that the outputs are always an initial subsequence of the inputs: in other words no data is lost and order is preserved. Condition 2 says that when a buffer is empty it must accept any input, and condition 3 says that a nonempty buffer cannot refuse to output the next item which is due. This specification has been well studied before, for example in [Roscoe1982] and [Brookes*et al*1984], and indeed the law established in [Roscoe1982] that, if P and Q are buffers then so is $P \gg Q$, proves B_M satisfies it.

[Roscoe1982] also proved that B_M equalled $B^M_{\langle\rangle}$, where the processes B^M_s for $s \in T^*$ with $|s| \leq M$ are defined as follows.

$$\begin{array}{rcll} B^M_{\langle\rangle} & = & in?x \rightarrow B^M_{\langle x\rangle} & \\ B^M_{s\langle y\rangle} & = & (in?x \rightarrow B^M_{\langle x\rangle s\langle y\rangle}) \mathbin{[\!]} (out!y \rightarrow B^M_s) & \text{if } |s| < M-1 \\ B^M_{s\langle y\rangle} & = & out!y \rightarrow B^M_s & \text{if } |s| = M-1 \end{array}$$

We are going to use these well-tried examples, namely proving B_M is a buffer and that it equals $B^M_{\langle\rangle}$, to illustrate the techniques we develop.

In order to be able to apply our results about fixed point induction we must find a recursive definition that is provably equivalent to the network or other CSP term we are considering. One convenient way of doing this, and the method we will concentrate on, is based on operational semantics.

The operational semantics of CSP [Brookes*et al*1986] describes how a CSP term is modified by rewrite rules as it performs various actions: essentially it provides an abstract interpreter for the notation. We can think of the finite or infinite set of terms which is reachable from a given term P as being the state-space of P. It will be denoted $S(P)$.

The set of all CSP terms with a given alphabet Σ forms a labelled transition system under the operational semantics — transitions being drawn from $\Sigma \cup \{\tau\}$, where τ represents internal, invisible actions. Clearly $S(P)$ is a sub-transition system for any term P, which cannot be distinguished from the whole set if we are only allowed to start our transitions from P.

Given any labelled transition system it is easy to define functions which construct the sets of finite or infinite traces (sequences of visible actions) an element can perform, as well as the set of divergences (traces on which the process can perform an infinite sequence of internal actions) and failures — pairs (s, X) such that the process can perform the finite trace s and then come into a stable state (one where no internal action is possible) where no element of X is possible. Provided we close up as appropriate under divergence in defining these functions, we can form abstraction functions which map the transition system into any of the CSP models based on these types of behaviour. As a consequence of the congruence theorems of [Brookes*et al*1986] and [Roscoe1988b] we know that the result of doing this to an element of the transition system of CSP terms formed by the operational semantics is exactly the same value as predicted in the appropriate denotational semantics (i) in the case where we are using finitely nondeterministic CSP and the failures divergence model and (ii) where we are allowed to use unboundedly nondeterministic constructs but use the failures/divergences/infinite traces model. Of course, exactly the same statement holds good of a CSP process P when we think of P as an element of the transition system $S(P)$ rather than the the transition system of the whole language.

We will concentrate on the failures/divergence model and only allow finitely nondeterministic CSP. In particular this means that we must restrict ourselves to hiding only finite sets of events. On the positive side it means we can use the standard version $\mathcal{N}$ of the failures/divergences model with the compactness axiom

$$(\forall Y \in \wp(X).(s,Y) \in F) \Rightarrow (s,X) \in F$$

which improves the topological properties of the model (see [Roscoe1988a] for details of what happens without this axiom). ($\wp(X)$ is the set of all finite subsets of X.) Under the definedness order $\leq$ (introduced in [Roscoe1988a]), this model has as its maximal elements precisely the divergence-free processes.

Suppose $\Phi(\cdot)$ is the abstraction map that maps an arbitrary finitely nondeterministic transition system into $\mathcal{N}$. It is easy to see that, for any $P \in T$, we have

$$\Phi(P) = x : P^0 \rightarrow \sqcap\{\Phi(P') \mid P \stackrel{x}{\longrightarrow} P'\}$$

if P is stable in the sense discussed above. If P is not stable, we get instead

$$\Phi(P) = (x : P^0 \rightarrow \sqcap\{\Phi(P') \mid P \stackrel{x}{\longrightarrow} P'\}) \rhd \sqcap\{\Phi(P') \mid P \stackrel{\tau}{\longrightarrow} P'\}$$

where $P \rhd Q$ is a shorthand for $(P \,[\!]\, Q) \sqcap Q$, and P^0 is the set of visible actions P can perform on its first step. (The uses of the $\sqcap$ operation are well defined because the sets over which they have been applied are always nonempty and finite.)

If we replace all occurrences of $\Phi(Q)$'s in the above equations by V_Q, for a process variable V_Q (Q varies over T), we get a mutual recursion defined over vectors of processes indexed by the transition system T. We know from the above that the vector of $\Phi(P)$ as P varies is one solution to this recursion, which we will henceforth refer to as the *state recursion*. In fact, it is the least solution, which is a corollary to part of the proof of the congruence theorem in [Roscoe1988b][9]. The assumptions we have made about finite nondeterminism ensure that the function from $\mathcal{N}^{S(P)}$ to itself associated with the state recursion is continuous (like all functions definable in this dialect of CSP).

In the case where the transition system T is the one produced by the operational semantics (on the whole set of CSP terms or on an $S(P)$), we know by the congruence theorem that the vector of $\Phi(Q)$'s is identical to the vector of denotational semantic values. Thus, given an arbitrary CSP term P, we have constructed an indexing set — $S(P)$ — and a mutual recursion over vectors indexed by it whose least solution has a component equal to P. We have thus achieved one part of what we set out to do in this section: to synthesize a recursion from a process term. We can now use this recursion and our fixed-point induction rules to prove properties of $S(P)$. It is worth making two observations about the state recursion.

- It will almost always be non-constructive, because any internal action causes an unguarded recursive call. It will, however, always be nondestructive since it is formed purely from nondestructive operations (the two forms of choice and prefixing).
- The particularly simple form of the recursion is likely to make the checking of individual components of $R(\underline{V}) \Rightarrow R(F(\underline{V}))$ easy. However there may be a large finite number, or even an infinite number, of such components.

Because of the first of these observations and the fact that the failures/divergence model $\mathcal{N}$ has no top element, only two of the versions of recursion induction discussed in the last section are potentially applicable. Scott induction can be used to prove properties such as $R(\underline{V}) \equiv \forall Q. V_Q \sqsubseteq C_Q$ for any vector $\underline{C}$ of processes. (Here, $\sqsubseteq$ is the standard nondeterminism order on $\mathcal{N}$.) This is not the sort of property which is very useful to have if we are seeking to use (a component of) the vector $\underline{V}$, but it may well be useful if we are seeking to use $\underline{C}$. In other words, we can prove that one vector refines

[9] That reference constructs a sequence of approximate abstraction functions Φ_α which are just the iterations of the above recursion, and shows that this sequence eventually reaches the usual abstraction function Φ.

another (which can be thought of as the specification) by doing induction on the specification.

The other form of induction which is applicable, and the one we shall concentrate on, is that established by Theorem 0.2.2. Of course we will use the stronger definedness order on CSP along with this, since it gives us far more maximal elements. We will, however, have to assume that our process P has been proved divergence free before we start this part of our analysis. It should be easy to see that this implies that all elements of $S(P)$ are also divergence free and hence that the vector consisting of the semantic values of all of these processes is maximal, under $\leq$, in the product space $\mathcal{N}^{S(P)}$.

It is certainly worth noting that, under this divergence freedom assumption, the vector of semantic values is not only the least, but actually the only, fixed point of the derived recursion. Thus if one can find any vector of values which is a fixed point, the two must be equal. This gives us a unique fixed point principle which can be used in place of the commonly used metric one which is applicable to constructive state-based mutual recursions. We will shortly apply a version of it to our buffer example.

The major obstacle to the application of the constructions given above is the observation made earlier about the number of cases that need to be checked. Realistically we can only hope to employ them (at least by hand) if we can find some reasonably concise classification of the states — and perhaps a useful notation for them — which will allow us to combine all of the cases into a few pieces of analysis. From now on we will restrict ourselves to a particular class of CSP processes, one which is interesting in itself and which will allow us to introduce some specific ideas to help codify and reduce this complexity.

This class will be the parallel compositions of a number of deterministic processes with a single hiding operation at the outside. Usually we can expect this hiding to conceal the internal communications of the system, leaving visible just those ones which are only in the alphabet of a single process. As has been observed elsewhere (e.g. [Roscoe and Dathi1988]) this syntax can, using the various laws of CSP and suitable renaming of hidden communications, cover any composition of (in this case deterministic) processes using interleavings of the parallel and hiding operators. For any hiding that takes place at inner levels can be moved to the outside by means of the law

$$(P \backslash X \parallel Q) = (P \parallel Q) \backslash X \quad \text{if } X \cap \alpha Q = \emptyset$$

after renaming of the hidden communications which appear in αQ. This type of example shows up our difficulty with number of states very well, since even if all of the processes we are combining in parallel are finite state (say with the same number of states) then the potential number of states of the combination grows exponentially with the number of processes.

The advantage of this type of process from our point of view is that its structure is static under the operational semantics. In other words if the initial state is $(P_1 \parallel \cdots \parallel P_n)\backslash X$, then every element of $S(P)$ has the form $(P_1' \parallel \cdots \parallel P_n')\backslash X$ where each P_i is deterministic and the alphabet of P_i' is the same as that of P_i. We will write Σ_i for the alphabet of the ith component of the parallel and $\Sigma = \bigcup\{\Sigma_i \mid 1 \leq i \leq n\}$, the alphabet of the system before hiding.

Even though the processes P_i we are composing together are assumed to be deterministic, this does not mean that they are free of internal (τ) actions. Indeed, because of the operational semantics of recursion it is impossible to build a nonterminating CSP term whose standard operational semantics is free of τs. These internal τs will almost always be an inconvenience, since they add to the size and complexity of the state-space without, in this instance, adding anything to the semantics. The analysis of our restricted class of processes would be easier if we could think of the P_i as generating only visible actions, so we could restrict our attention to how the parallel and hiding operators combine these and turn some of them into τs.

It is easy, for each alphabet Σ, to construct a transition system with one element for each deterministic process with alphabet Σ and where there are no τ actions. If we think of the deterministic processes as prefix-closed nonempty sets of traces, we will have $P \xrightarrow{a} Q$ if and only if $\langle a\rangle \in P$ and $Q = \{s \mid \langle a\rangle s \in P\} = P/a$, say. The abstraction map $\Phi(\cdot)$ applied to this system will produce exactly the values we would hope. And we note that because all states in this transition system are now stable and there is never more than one arc with a given label from any node, the states all satisfy the equation

$$\Phi(P) = x : P^0 \rightarrow \Phi(P/a)$$

In determining the state space of our system, and thus in subsequent computations, we can substitute objects with this nicer transition behaviour for the standard operational semantics of the deterministic processes without destroying any of the analysis carried out earlier. In deriving the transitions of the network $(P_1 \parallel \cdots \parallel P_n)\backslash X$ from the transitions of the P_i, we will assume that the P_i are drawn from the canonical systems of deterministic processes (with appropriate alphabets). This substitution is easily justified in terms of the work in [Brookes*et al*1986] and [Roscoe1988b] where *operational environments* are used. The congruence theorems proved in those papers extend to the cases where the CSP terms we consider have free variables, and the values of those free variables are determined by environments which assign elements of transition systems to the variables.

With this simplification the state-space $S((P_1 \parallel \cdots \parallel P_n)\backslash X)$ of the system is now a set of objects of the form $N\backslash X = ({P'}_1 \parallel \cdots \parallel {P'}_n)\backslash X$ where the P_i' are drawn from the canonical deterministic transition systems. The transitions of these states, and hence the mutual recursion derived from them, are easy to deduce from those of the network N before hiding. For a given element $N\backslash X = (Q_1 \parallel \cdots \parallel Q_n)\backslash X$ of the state space there is exactly one transition for any $a \in \Sigma$ such that $\forall i.a \in \Sigma_i \Rightarrow a \in Q_i^0$, and then

$$\begin{array}{ll} N\backslash X \xrightarrow{a} N'\backslash X & \text{if } a \notin X \\ N\backslash X \xrightarrow{\tau} N'\backslash X & \text{if } a \in X \end{array}$$

where $N' = (Q_1' \parallel \cdots \parallel Q_n')$, for $Q_i' = Q_i/a$ if $a \in \Sigma_i$ and $Q_i' = Q_i$ otherwise. In either case we have $N \xrightarrow{a} N'$. Clearly the state is stable if and only if there is no $a \in X$ such that $a \in N^0$.

The clauses of the recursion as the state varies can now be derived from our earlier discussion. We index the recursion by the N as above.

$$\begin{array}{rcll} V_N & = & a : N^0 \to V_{N/a} & \text{if } N^0 \cap X = \emptyset \\ & = & a : (N^0 - X) \to V_{N/a} & \\ & & \quad \rhd \bigsqcap \{V_{N/a} \mid a \in N^0 \cap X\} & \text{otherwise} \end{array}$$

Now, on the assumption that our original network $(P_1 \parallel \cdots \parallel P_n)\backslash X$ was divergence free, we know that the vector $\underline{V}$ of semantic values of these states is the unique (and maximal in the product space) fixed point of this recursion. This result can be used directly, since by demonstrating that some other vector is a fixed point we can prove it equals $\underline{V}$. It can also be used, via Theorem 0.2.2, to prove results about $\underline{V}$ by fixed point induction.

There is another way one can derive the state recursion of one of our parallel/hiding combinations without thinking about operational semantics at all. We have already observed that any deterministic process can be written uniquely in the form

$$a : A \to P(a)$$

where each $P(a)$ is deterministic. If we write out all of the P_i' in a typical state $({P'}_1 \parallel \cdots \parallel {P'}_n)\backslash X$ in this way, then we can apply (i) the *expansion* laws of $\|$ to bring one step of communications out of the parallel operator and then (ii) one of the laws

$$\begin{array}{lcll} (a : A \to P(a))\backslash X & = & a : A \to (P(a)\backslash X) & \text{if } X \cap A = \emptyset \\ (a : A \to P(a))\backslash X & = & a : (A - X) \to (P(a)\backslash X) & \\ & & \quad \rhd \bigsqcap\{P(a)\backslash X \mid a \in A \cap X\} & \text{otherwise} \end{array}$$

This will derive exactly the same equation among the semantic values of our state space as was produced by the constructions above.

We will now return to our example of M one-place buffers piped together. This example is not actually in exactly the same form as the one we have been considering, since instead of using parallel and hiding operators separately we have used the hybrid operator $\gg$. The main consequence of this is that the hiding gets done at intermediate stages as we put the network together rather than all at once at the end. However, with some renaming of internal communications and applications of the law mentioned above, this system becomes equal to

$$(B_c^{in} \parallel B_d^c \parallel \cdots \parallel B_{out}^h)\backslash X$$

where $c, d, \ldots, h$ are a sequence of distinct channel names we have picked different from *in* and *out*, $X = c.T \cup \cdots \cup h.T$ and

$$B_b^a = a?x : T \to b!x \to B_b^a$$

In fact, not only the denotational semantics, but also the transitions of the operational semantics, of this system are identical to that of the original one

$$B_1 \gg B_1 \gg \cdots \gg B_1$$

so that in fact by exploring the state-space of one system we are exploring that of the other. The state-spaces of these systems are particularly easy to describe: each component process either holds a value $x \in T$ or is empty, and furthermore any combination of these is possible. A typical state can thus be denoted by an M-tuple of elements of $T \cup \{\epsilon\}$, where ϵ or $x \in T$ in the ith place denotes that the ith process is empty or contains x. The transitions of the state $\sigma \in (T \cup \{\epsilon\})^M$ (a set which we will abbreviate as S) are precisely those described below.

- If $\sigma = \langle\epsilon\rangle\sigma'$, then there is an action labelled $in.x$ for every $x \in T$ to the state $\langle x\rangle\sigma'$.

- If $\sigma = \sigma'\langle x\rangle$ for $x \in T$, then there is an action labelled $out.x$ to the state $\sigma'\langle\epsilon\rangle$.
- If $\sigma = \sigma_1\langle x, \epsilon\rangle\sigma_2$, then there is a τ action to the state $\sigma_1\langle\epsilon, x\rangle\sigma_2$ corresponding to the communication from one element of the chain to the next.

Thus a state is stable precisely when no ϵ follows an element of T in the M-tuple.

We can tabulate the various cases of the state recursion, defining a process V_σ for every state of the type discussed above. The stable cases first. If $\sigma = \langle\epsilon, \epsilon, \ldots, \epsilon\rangle = \epsilon^M$, we get

$$V_\sigma = in?x : T \rightarrow V_{\langle x\rangle\epsilon^{M-1}}$$

If $\sigma = \epsilon^k\sigma'\langle y\rangle$, where $k > 0$ and σ' contains no ϵ, then

$$V_\sigma = (in?x : T \rightarrow V_{\langle x\rangle\epsilon^{k-1}\sigma'\langle y\rangle}) \mathbin{[\!]} (out!y \rightarrow V_{\epsilon^k\sigma'\langle\epsilon\rangle})$$

If $\sigma = \sigma'\langle y\rangle$ contains no ϵ, then

$$V_\sigma = out!y \rightarrow V_{\sigma'\langle\epsilon\rangle}$$

Secondly, the unstable cases. An unstable state of the form $\langle x\rangle\sigma'\langle\epsilon\rangle$ has no external actions, so we get

$$V_\sigma = \bigsqcap\{V_{\sigma^*} \mid \sigma \xrightarrow{\tau} \sigma^*\}$$

One of the form $\langle\epsilon\rangle\sigma'\langle\epsilon\rangle$ can input, so we get

$$V_\sigma = (in?x : T \rightarrow V_{\langle x\rangle\sigma'\langle\epsilon\rangle}) \rhd (\bigsqcap\{V_{\sigma^*} \mid \sigma \xrightarrow{\tau} \sigma^*\})$$

One of the form $\langle z\rangle\sigma'\langle y\rangle$ can perform an output, so we get

$$V_\sigma = (out!y \rightarrow V_{\langle z\rangle\sigma'\langle\epsilon\rangle}) \rhd (\bigsqcap\{V_{\sigma^*} \mid \sigma \xrightarrow{\tau} \sigma^*\})$$

And finally, one of the form $\langle\epsilon\rangle\sigma'\langle y\rangle$ can do either:

$$\begin{aligned} V_\sigma \;\; = \;\; & (in?x : T \rightarrow V_{\langle x\rangle\sigma'\langle y\rangle}) \mathbin{[\!]} (out!y \rightarrow V_{\langle\epsilon\rangle\sigma'\langle\epsilon\rangle}) \\ & \rhd (\bigsqcap\{V_{\sigma^*} \mid \sigma \xrightarrow{\tau} \sigma^*\}) \end{aligned}$$

A prerequisite of using our methods on this system is proving it free of divergence. This is an easy consequence of the following rule, which applies to any system of the form $(P_1 \parallel P_2 \parallel \cdots \parallel P_n)\backslash X$, where X consists of all communications that are in the alphabets of at least two of the P_i. Versions of this rule have previously appeared in several references, for example [Roscoe1982, Dathi1989].

Lemma 3.1 *Suppose we can place a partial order between the nodes P_i such that (i) each process is comparable in the order with all its neighbours (i.e. P_j such that $\alpha P_i \cap \alpha P_j \neq \emptyset$) and (ii) no process communicates infinitely with neighbours less than itself without either communicating externally or with a neighbour greater than itself. Then the system is divergence-free.*

The partial order we use on the current system is the obvious linear order where the leftmost process is the least.

We can thus deduce that the state-recursion has a unique and maximal fixed point. This fact can be used to prove that, for all states σ, we have

$$V_\sigma = B^M_{|\sigma|}$$

where B^M_s is as described earlier and $|\sigma|$ is the σ with all ϵ's removed (i.e. the actual contents of the multi-stage buffer). For consider the vector $\underline{W}$, indexed by S, where $W_\sigma = B^M_{|\sigma|}$. It is easy to show that this is a fixed point of the state recursion. For example, no internal action changes $|\sigma|$, and so all of the unstable clauses of the application of the function F associated with the recursion have the form

$$F(\underline{W})_\sigma = (x : A \rightarrow B^M_{f(\sigma,x)}) \rhd B^M_{|\sigma|}$$

where in every case $A \subseteq (B^M_{|\sigma|})^0$ and, for all $a \in A$, $B^M_{f(\sigma,a)} = B^M_{|\sigma|}/a$. The fact that these clauses all work thus follows from the fact that, if $A \subseteq B$ and $\forall a \in A.P(a) \sqsupseteq Q(a)$, then

$$(a : A \rightarrow P(a)) \rhd (b : B \rightarrow Q(b)) = (b : B \rightarrow Q(b))$$

After a little more analysis of this type we would conclude that the two vectors $\underline{V}$ and $\underline{W}$ are equal, so that in particular the network B_M equals the process $B^M_{\langle\rangle}$ as claimed earlier. We will show shortly how this type of argument can be neatly summarised in an easily-stated principle.

We may well want to prove more abstract properties of our system than equality with some other process, and besides it may not always be as easy as it was here to find an alternative description of the processes represented by the state space. As an example to show how to prove such properties, we will show how the recursion derived from the network B_M can be used to prove directly that it is a buffer in the sense described earlier.

We need a predicate which applies to the whole product space $\mathcal{N}^S$, which is closed in a compact topology, which is satisfiable, which inducts through the state-recursion, and which implies that the ϵ^M-state is a buffer. The property we will use is

$$R(\underline{W}) \equiv \forall\sigma.(W_\sigma \text{ is a } |\sigma|\text{-buffer})$$

where, for $w \in T^*$, a w-buffer is a divergence-free process $\langle F, \emptyset\rangle$ such that, for all $(s, X) \in F$,

$$\begin{aligned} s{\downarrow}out &\leq w\,(s{\downarrow}in) && (4)\\ s{\downarrow}out = w\,(s{\downarrow}in) &\Rightarrow X \cap in.T = \emptyset && (5)\\ (s{\downarrow}out)\langle a\rangle \leq w\,(s{\downarrow}in) &\Rightarrow out.a \notin X && (6) \end{aligned}$$

A w-buffer is simply a buffer that initially contains the sequence w. Obviously, a $\langle\,\rangle$-buffer is precisely a buffer in the sense that we specified earlier.

One can put topologies on $\mathcal{N}$ and its product spaces in several different ways, though when one is considering (as we are, for reasons discussed earlier) the case of finite alphabets, several of these actually coincide. The author introduced a sequence of topologies termed (α), (β), (γ) and (δ) in [Roscoe1988a], which coincide in the case of finite alphabets and finite product spaces. The α-topology is just the topology defined, as described in the last section, with respect to the standard length-of-trace metric. The others were defined as follows, over the product space $\mathcal{N}^\Lambda$. (Below, the failures or divergences of the λ-component of the vector $\underline{P}$ are denoted D^P_λ or F^P_λ respectively.)

(β) A set C is closed iff whenever $\underline{P} \notin C$ then there are finite sets T of traces and Ψ of indices such that, for any $\underline{Q}$, if

$$s \in D^P_\lambda \Leftrightarrow s \in D^Q_\lambda \text{ and } \{X \mid (s,X) \in F^P_\lambda\} = \{X \mid (s,X) \in F^Q_\lambda\}$$

for all $s \in T$ and $\lambda \in \Psi$, then $\underline{Q} \notin C$.

(γ) C is closed iff whenever $\underline{P} \notin C$ then there are finite sets Ψ of indices, T of traces and F of failures such that, for any $\underline{Q}$, if whenever $\lambda \in \Psi$

$$s \in D^P_\lambda \Leftrightarrow s \in D^Q_\lambda$$

for all $s \in T$ and

$$(s,X) \in F^P_\lambda \Leftrightarrow (s,X) \in F^Q_\lambda$$

for all $(s,X) \in F$, then $\underline{Q} \notin C$.

(δ) C is closed iff whenever $\underline{P} \notin C$ then there are finite sets Ψ of indices, T of traces and F of *finite* failures (failures (s, X) with X finite) such that, for any $\underline{Q}$, if whenever $\lambda \in \Psi$

$$s \in D_\lambda^P \Leftrightarrow s \in D_\lambda^Q$$

for all $s \in T$ and

$$(s, X) \in F_\lambda^P \Leftrightarrow (s, X) \in F_\lambda^Q$$

for all $(s, X) \in F$, then $\underline{Q} \notin C$.

These topologies are successively weaker. The following result from [Roscoe1988a] lists some of their basic properties.

LEMMA 3.2 *1. The conditions β, γ and δ are equivalent if the underlying alphabet Σ is finite. If it is infinite δ is strictly stronger than γ and γ is strictly stronger than β. α is equivalent to β if the alphabet is finite and the index-set Λ is finite. Otherwise β is strictly stronger than α.*

2. *For all $\xi \in \{\beta, \gamma, \delta\}$ the ξ-topology on $\mathcal{N}^\Lambda$ defined above is well-defined. The topology generated by ξ on $\mathcal{N}^\Lambda$ is the same as the product topology where each copy of $\mathcal{N}$ is given the topology generated by ξ.*

3. *The δ-topology on $\mathcal{N}^\Lambda$ is compact, Hausdorff and zero-dimensional.*

4. *If $\underline{P} \in \mathcal{N}^\Lambda$, each of the sets $\{\underline{Q} \mid \underline{Q} \geq \underline{P}\}$, $\{\underline{Q} \mid \underline{Q} \leq \underline{P}\}$, $\{\underline{Q} \mid \underline{Q} \sqsupseteq \underline{P}\}$ and $\{\underline{Q} \mid \underline{Q} \sqsubseteq \underline{P}\}$ are closed in the δ-topology (and hence in the others as well).*

Since in our case the set T of values passed over channels is finite, we know the state space S is as well and hence all of these topologies coincide. But in fact the property of being a w-buffer is (δ)-closed even in the general case, since any failure shows up in a process $\langle F, D\rangle$ in one of

1. a failure $(s, \emptyset)$ where s violates (a), or
2. a failure $(s, \{in.x\})$ where $s{\downarrow}out = w\,(s{\downarrow}in)$, or
3. a failure $(s, \{out.a\})$ where $(s{\downarrow}out)\langle a\rangle \leq w\,(s{\downarrow}in)$

even though, as stated, the predicate is phrased in terms of infinite as well as finite failures. Since the set of processes satisfying R is just the Cartesian product of (δ)-closed sets, it follows by Lemma 0.3.2(2) that the set of vectors satisfying R is closed in the product topology.

R is satisfiable, since for any sequence $w = \langle a_1, \ldots, a_r \rangle$ the process

$$out.a_1 \rightarrow \cdots \rightarrow out.a_r \rightarrow B_1$$

is easily shown to be a w-buffer.

All we now have to do is prove that $\forall \underline{W}.R(\underline{W}) \Rightarrow R(F(\underline{W}))$, where F is the function of our state recursion. Much as in the case of the equality proof above, this simply requires some easy case-checking, which is assisted by the following observations.

1. Any behavioural specification like w-buffer is distributive, in the sense that if U is a finite set of w-buffers, then $\bigsqcap U$ is one too.

2. If P and Q are divergence-free then so is $P \rhd Q$, and its failures are $F_Q \cup \{(s, X) \in F_P \mid s \neq \langle\rangle\}$. In particular this implies that, if $a : A \rightarrow P(a)$ and Q are both w-buffers, and $B \subseteq A$, then $(a : B \rightarrow P(a)) \rhd Q$ is a w-buffer.

The details of the proof are left to the reader.

We took care above to show that our predicate R was δ-closed, so that Theorem 0.2.2 would be applicable even if the alphabet were infinite. In fact, because of the particular structure of the CSP model $\mathcal{N}$, it turns out that, for continuous functions like our state recursion (and all functions definable using this CSP dialect), closure in the β-topology is enough. This is proved in [Roscoe1988a], via a more detailed analysis of the way the iteration from bottom converges to the least fixed point. One practical benefit of this is that behavioural predicates on $\mathcal{N}$ are always γ-closed, and hence β-closed, but may not be δ-closed if the alphabet is infinite.

The δ-topology actually coincides with another well-known order topology, the *Lawson* topology (or Λ-topology). It can be defined to be the coarsest common refinement of the two T_0 topologies we have already met, namely the Scott topology and the lower topology. For detailed discussion of this topology, see [Gierz*et al*1980]. For a consistently complete, algebraic[10] cpo

[10] An algebraic cpo is one where the set of *finite* elements less than or equal to x is directed with limit x. A finite element is one e such that, whenever Δ is directed with $\bigsqcup \Delta \geq e$, then $\exists y \in \Delta . y \geq e$. The failures/divergence model $\mathcal{N}$ is algebraic and consistently complete under both $\leq$ and $\sqsubseteq$.

it is always compact Hausdorff. If the alphabet is finite then the δ-topology (and hence the others as well, as described in Lemma 0.3.2), is equivalent to the Lawson topology with respect to both orders. If the alphabet is infinite then it is the same as the $\sqsubseteq$-Lawson topology. The $\leq$-Lawson topology is then different from all the others we have met. The relationship of these two compact Hausdorff topologies will be an interesting topic for further work: it may be possible to take advantage of them in combination just as we did with the pair of orders.

4 Conclusions

In this paper we have seen how convergence links topology and computer science, and have in particular studied conditions under which fixed point induction is valid.

In Section 0.3 we showed how to take advantage of the topological structure of the CSP model $\mathcal{N}$ and construct a sequence which converged on (the state-vector of) an arbitrary process, and then use fixed point induction to establish properties of the process itself. While we showed how the state-recursion could be derived for an arbitrary CSP term, we concentrated on parallel/hiding combinations of deterministic processes. The method we evolved for proving things about the process using this recursion can be summarised in the following general principle.

> STATE INDUCTION Suppose we have a divergence-free CSP process with state-space $S(P)$, and to each $Q \in S(P)$ we associate a satisfiable and β-closed predicate R_Q. Then if we can prove each $R_Q(Q)$ holds simply on the assumption that $R_{Q'}(Q')$ holds of all states Q' reachable in one step from Q using the Q-clause of the state-recursion, we can infer $\forall Q \in S(P).R_Q(Q)$.

One particular example we saw this being applied to was where each R_Q specified equality with a particular process W_Q. Equally valid would be the case where each R_Q specified that the Q-component refined (in either order) W_Q. Thus, for example, if P is divergence-free and we can prove that $F(\underline{W}) \sqsupseteq \underline{W}$, then the state-vector of P also refines $\underline{W}$. (Since we have used the definedness order $\leq$ crucially in proving this, it provides an interesting case of the two orders complementing each other.)

We have seen how the state-recursion technique applies to one fairly simple and regular example. The combinatorial explosion which arises from parallel composition means that we are only likely to be able to apply the techniques by hand to examples where there is enough regularity to allow us to describe the state-space reasonable compactly, and to find a classification of states which allows us to deal with only a few essentially different cases. The work we have described here is clearly closely related to the very active field of exploring state spaces mechanically, known as *model checking*. Some references to this are [Browne*et al*1986, Burch*et al*1990, Kurshan and McMillan1989]. The work we have carried out here could, indeed, be viewed as providing the mathematical basis for using model-checking techniques for proving properties of CSP programs specified over $\mathcal{N}$.

References

[America and Rutten1988] America, P., and Rutten, J. (1988). *Solving reflexive domain equations in a category of complete metric spaces*. Proceedings of MFPLS87, Springer LNCS 298 pp 254-288.

[Brookes1983] Brookes, S. D. (1983). *A model for communicating sequential processes*. Oxford University D.Phil. thesis.

[Brookes*et al*1984] Brookes, S. D., Hoare, C. A. R., and Roscoe, A. W. (1984). *A theory of communicating sequential processes*. Journal of the Association for Computing Machinery, **31**, 3, pp 560-599.

[Brookes and Roscoe1985] Brookes, S. D., and Roscoe A. W. (1985). *An improved failures model for communicating processes*. Springer LNCS 197, pp 281-305.

[Brookes*et al*1986] Brookes, S. D., Roscoe A. W., and Walker, D. J. (1986). *An operational semantics for CSP*. Submitted for publication.

[Browne*et al*1986] Browne, M. C., Clarke, E. M., and Grumberg, O. (1986). *Reasoning about networks with many identical finite state processes*. Proc. 5th ACM Symposium on Principles of Distributed Computing.

[Burch*et al*1990] Burch, J. R., Clarke, E. M., Dill, D. L., and Hwang, L. J. (1990) *Symbolic model checking:* 10^{20} *states and beyond*. Proc. 5th IEEE Annual Symposium on Logic in Computer Science, IEEE Press.

[Dathi1989] Dathi, N. (1989). *Deadlock and deadlock-freedom*. Oxford University D.Phil. thesis.

[Gierz*et al*1980] Gierz, G., Hofmann, K. H., Keimel, K., Lawson, J. D., Mislove, M., and Scott, D. S. (1980). *A compendium of continuous lattices*, Springer-Verlag.

[Hoare1985] Hoare, C. A. R. (1985) *Communicating sequential processes*. Prentice-Hall International.

[Hoare*et al*1981] Hoare, C. A. R., Brookes, S. D., and Roscoe, A. W. (1981). *A theory of communicating sequential processes*, Oxford University Computing Laboratory, Programming Research Group, Technical Report PRG-16.

[Johnstone1982] Johnstone, P.T. (1982) *Stone spaces*. Cambridge University Press.

[Kurshan and McMillan1989] Kurshan, R. P., and McMillan, K. (1989). *A structural induction theorem for processes*. Proc. 8th ACM Symposium on Principles of Distributed Computing, Edmonton.

[Nelson1987] Nelson, G. (1987). *A generalisation of Dijkstra's calculus*. Research report 16, Digital Systems Research Center.

[Reed1988] Reed, G. M. (1988). *A uniform mathematical theory for real-time distributed computing*. Oxford University D.Phil. thesis.

[Reed and Roscoe1986] Reed, G. M., and Roscoe, A. W. (1986). *A timed model for communicating sequential processes*. Proceedings of ICALP'86, Springer LNCS 226, pp 314-323.

[Reed and Roscoe1988] Reed, G. M., and Roscoe, A. W. (1988). *Metric spaces as models for real-time concurrency*. Proceedings of MFPLS87, Springer LNCS 298 pp 331-343.

[Roscoe1982] Roscoe, A. W. (1982). *A mathematical theory of communicating processes*. Oxford University D.Phil. thesis.

[Roscoe1988a] Roscoe, A. W. (1988a). *An alternative order for the failures model*. in 'Two Papers on CSP', Technical Monograph PRG-67, Oxford University Computing Laboratory.

[Roscoe1988b] Roscoe, A. W. (1988b). *Unbounded nondeterminism in CSP*. in 'Two Papers on CSP', Technical Monograph PRG-67, Oxford University Computing Laboratory.

[Roscoe and Dathi1988] Roscoe, A. W. and Dathi, N. (1987). *The Pursuit of deadlock freedom.* Information and Computation **75**, 3.

[Roy1969] Roy, P. (1969). *Failure of equivalence of dimension concepts for metric spaces.* Bull. Amer. Math. Soc. **68** pp609-613.

[Schneider1990] Schneider, S. (1990) *Correctness and communication in real-time systems* Oxford University D. Phil Thesis.

[Smyth1988] Smyth, M. B. (1988). *Quasi-uniformities: reconciling domains with metric spaces.* Proceedings of MFPLS87, Springer LNCS 298 pp236-253.