The relationship between CSP, FDR and Büchi automata

A.W. Roscoe and Thomas Gibson-Robinson
Department of Computer Science, University of Oxford, Parks Road, Oxford OX1 3QD, U

August 11, 2016

Abstract

Two long standing approaches to specifying and verifying properties of finite-state systems are Büchi automata, which are specialised for reasoning about infinite traces, and the combination of CSP and its refinement checker FDR, which offer some scope for reasoning about infinite traces in addition to capturing a wide variety of finitely observable behaviour. In this paper we demonstrate that the infinite trace properties that the long-standing functionality of FDR can decide are exactly the same as deterministic Büchi automata. We also show how a simple extension of FDR allows it to efficiently verify general Büchi automata and consequently LTL formulae and properties under fairness.

1 Introduction

In this paper we explore the relationship between two ways of specifying and verifying behavioural properties of finite-state systems. The first of these is the use of *refinement* between a specification process and a distributive context over the CSP denotational models as supported by the well-known refinement checker FDR [10, 3]. The second is the use of Büchi automata, which are the key to verifying LTL properties in a number of model checkers [16]. All these terms will be defined in this paper. We will sometimes abbreviate "Büchi automata/on" as *Büchi*.

A *behavioural property* is a property of a process P which holds if and only if all P's observable individual behaviours (i.e. linear observations) satisfy some fixed property. Büchi and the most common mode of using FDR (proving of a process P that it refines some finite-state specification process S) both represent sorts of behavioural specifications. However the CSP/FDR approach deals mainly with *finitary* properties (properties of finitely observable behaviours) while Büchi are representations only of infinitary properties, in the sense that they refer only to infinite traces.

In this paper we concentrate on behavioural properties. We show that every behavioural infinite trace property that FDR can naturally[1] express can be reduced to the form $C[P]$ *is divergence-free* where C is a stylised CSP context,

[1] [6] shows that one can go beyond this by putting the process that one wants to check in a context on the LHS of a refinement check rather than the usual, and much more efficient, RHS. We will discuss the consequences of this later.

and we show that these properties are exactly those characterised by deterministic Büchi. A corollary to this is that FDR cannot at present efficiently decide properties represented by the more general nondeterministic Büchi automata, but we show how it can be extended to do so (based, as in other tools, on the *negation* of the intended Büchi.

In order to show the above results, we show how to factor a general CSP refinement check over a denotational model that includes divergence into two checks: one of the form $Spec \sqsubseteq_X P$ where X is a finite observation model of CSP; and one that its infinite traces satisfy a deterministic Büchi.

The rest of the paper is structured as follows. In the next section we present background: a summary of CSP, its abstract semantic models and its operational semantics, and basic facts about Büchi automata. It also analyses the operational semantics of so-called *distributive* contexts in preparation for later sections. Section 3 analyses the capability of CSP and FDR to express behavioural properties, and shows how these can be factored into a finite-behaviour check of the form $Spec \sqsubseteq_X P$ and a divergence check. Section 4 shows how these divergence checks correspond to deterministic Büchi, whilst Section 5 shows how a simple extension to FDR can decide whether the infinite traces of a finite-state process satisfy a general Büchi, and report on the performance of our implementation.

2 Background

2.1 CSP and its semantics

CSP is based on instantaneous actions handshaken between a process and its environment, whether that environment consists of processes it is interacting with or some notional external observer. It enables the modelling and analysis of patterns of interaction. The books [4, 11, 13, 15] all provide thorough introductions to CSP. The main constructs that we will be using in this paper are set out below.

- The processes $STOP$ and **div** respectively do nothing, and diverge by repeating the internal action τ. Run_A and $Chaos_A$ can each perform any sequence of events from A, but while Run_A always offers the environment every member of A, $Chaos_A$ can nondeterministically choose to offer just those members of A it selects, including none at all. In the absence of the subscripts, the whole alphabet is assumed.

- $a \rightarrow P$ *prefixes* P with the communication a belonging to the set Σ of normal visible communications. Similarly $?x : A \rightarrow P(x)$ offers the choice of events in A and then behaves accordingly.

- CSP has several *choice* operators. $P \Box Q$ offers the environment the first visible events of P and Q, and $P \sqcap Q$ nondeterministically makes a decision via τ actions whether to behave like P or Q.

- $P \setminus X$ (hiding) behaves like P except that actions in X become τs.

- $P[\![R]\!]$ (renaming) behaves like P except that when P performs an action a, the new process performs some b that is related to a under the relation R.

- $P \underset{A}{\parallel} Q$ is a *parallel* operator under which P and Q act independently except that they have to agree (i.e. synchronise or handshake) on all communications in A. A number of other parallel operators can be defined in terms of this.

Other CSP operators such as $P;\ Q$ (sequential composition), $P \triangle Q$ (interrupt) $P \Theta_A Q$ (throw) and $P \rhd Q$ (asymmetric choice) do not play a direct role in this paper. However including them does not alter our results. However, we specifically exclude the *priority* operator introduced at the end of [13] because it radically changes CSP's operational and denotational semantics.

CSP has several styles of semantics that can be shown to be appropriately consistent with one another [11, 13]. The two styles that will concern us are *operational* semantics, in which rules are given that interpret any closed process term as a labelled transition system (LTS), and *behavioural* models, in which processes are identified with sets of observations that might be made from the outside.

An LTS models a process as a set of states that it moves between via actions in $\Sigma \cup \{\tau\}$, where τ cannot be seen or controlled by the environment. There may be many actions with the same label a single state, in which case the environment has has no control over which is followed.

The best known behavioural models of CSP are based on the following. *Traces* are sequences of visible communications a process can perform. *Failures* are combinations (s, X) of a finite trace s and a set of actions that the process can refuse in a *stable* state reachable on s. A state is stable if it cannot perform τ. *Divergences* are traces after which the process can perform an infinite uninterrupted sequence of τ actions, in other words diverge. The models are then

- $\mathcal{T}$ in which a process is identified with its set of finite traces;
- $\mathcal{F}$ in which it is modelled by its (stable) failures and finite traces;
- $\mathcal{N}$ in which it is modelled by its sets of failures and divergences, both extended by all extensions of divergences: it is *divergence strict*.

Traces, failures and divergences are all observations that can be made of a process in *linear time*. As described in [13], there is a range of other models based on other, usually richer, forms of linear observations.

Highly relevant to the present paper is the fact that traces can be extended to allow both finite and infinite cases. Whereas the the divergence-strict models without infinite traces (or similar) are only compositional for finitely nondeterministic CSP[2], models such as $\mathcal{T}^{\omega\Downarrow}$, the divergence-strict model with traces and infinite traces, are compositional for infinitely nondeterministic constructs as well. In this notation ω signifies the use of infinite traces, and $\Downarrow$ the fact that it is divergence strict. Thus $\mathcal{N} = \mathcal{F}^{\Downarrow}$.

Any node of an LTS can be interpreted as a member of any of these models, by simply recording the observations possible for the node of the chosen type(s), if necessary adding extra ones to allow for divergence strictness. The congruence between operational and denotational semantics is the result that, for the language interpreted as an LTS through the operational semantics, each closed

[2]All finite-state processes are finitely nondeterministic

term gives the same set of behaviours whether calculated by observing this LTS or by calculating the value directly over the denotational semantics appropriate to that the model being used. Such results are long established for the standard models of CSP and we take them as given.

In the present paper, because Büchi automata characterise properties of traces rather than more complex structures, we concentrate on CSP models and congruences that use only finite and infinite traces, and divergences, or at most involve one additional observation (a refusal set in the case of failures) at the *end* of finite traces.

Operational semantics and distributive contexts

The operational semantics of CSP have their origin in [2, 1]. As discussed in [13], they can be presented either in traditional SOS style or *combinator* style closely related to CSP's implementation in FDR. Some SOS operational semantic clauses are given below.

$$\frac{}{e \to P \xrightarrow{a} subs(a, e, P)}(a \in comms(e))$$

The rule above for prefix says what we might expect: that the initial events of $e \to P$ are $comms(e)$ and that the process then moves into the state where the effects of any inputs in the communication have been accounted for. The following rules describe how a parallel composition progresses: each side progresses independently except when they have to synchronise on a member of the interface set.

$$\frac{P \xrightarrow{\tau} P'}{P \underset{X}{\parallel} Q \xrightarrow{\tau} P' \underset{X}{\parallel} Q} \qquad \frac{Q \xrightarrow{\tau} Q'}{P \underset{X}{\parallel} Q \xrightarrow{\tau} P \underset{X}{\parallel} Q'}$$

There are three rules for ordinary visible events: two symmetric ones for $a \notin X$

$$\frac{P \xrightarrow{a} P'}{P \underset{X}{\parallel} Q \xrightarrow{a} P' \underset{X}{\parallel} Q}\ (a \in \Sigma \setminus X) \qquad \frac{Q \xrightarrow{a} Q'}{P \underset{X}{\parallel} Q \xrightarrow{a} P \underset{X}{\parallel} Q'}\ (a \in \Sigma \setminus X)$$

and one to show $a \in X$ requiring both participants to synchronise

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \underset{X}{\parallel} Q \xrightarrow{a} P' \underset{X}{\parallel} Q'}\ (a \in X)$$

Clauses for all the other operators can be found in [13, 11], for example.

It is well-known that other than recursion, every CSP operator is *distributive* in each argument, meaning that if F is a CSP operator, and S is any nonempty set of processes, $F(\sqcap S) = \sqcap\{F(P) \mid P \in S\}$: i.e. F distributes over nondeterministic choice. This is itself a corollary of the fact that over each model we consider, nondeterministic choice is represented as component-wise union of sets of behaviours, and every non-recursive operator is [13, Chapter 10] obtained by lifting a series of relations between individual linear behaviours of tuples of the arguments and behaviours of the combination. It follows that every CSP context $C[P]$ that is constructible without recursion and where P never appears

in two or more arguments of any operator[3] is also a distributive function of each CSP model.

In this paper we will understand the idea of *distributive context* syntactically: it will mean something of the above form. It is easy to show that such contexts satisfy the distributive condition, and therefore that for such a context $C[\cdot]$, and process *Spec*, the process P satisfies $Spec \sqsubseteq_X C[P]$ if and only if $P \sqsupseteq_X \bigsqcap\{Q \mid Spec \sqsubseteq_X C[Q]\}$, meaning that $Spec \sqsubseteq_X C[P]$ is a behavioural property[4].

It can be seen from the operational semantics of CSP that, if $C[\cdot]$ is a distributive context, then every state reachable from $C[P]$ is either: a process derived solely from $C[\cdot]$ (i.e. independent of P); or $C'[P']$ for a distributive context $C'[\cdot]$ applied to some state P' of P; or simply a state P' of P. Since in general C' can either be a constant context (one that ignores its argument) or the identity context, all three of these options reduce to the middle one: we can consider every state of $C[P]$ to be of the form $C'[P']$.

As there are no negative premises in CSP's operational semantics, if we compute the operational semantics of $C[Run]$ and $C[P]$ for any divergence-free process P, then whenever a state $C'[P']$ is reachable in the latter, then $C'[Run]$ is possible in the former and furthermore

- If $C'[P'] \xrightarrow{x} C''[P'']$ then either $x = \tau$ and $C' = C''$, or $C'[Run] \xrightarrow{x} C''[Run]$. The first of these cases corresponds to P' performing a τ which is promoted silently in the operational semantics of C' to take $C'[P']$ to $C'[P'']$ where $P' \xrightarrow{\tau} P''$. Indeed there is a relation R between traces of C's potential argument P and states $C'[\cdot]$ such that if P performs any trace s within $C[P]$ to move into the state P', then $C[P]$ can reach any of the states $C'[P']$ where s and C' are related by R.

- By our assumption about P being divergence free, there is only ever a finite chain of the first type of τ actions. In other words if $C'[P']$ can diverge immediately, so can $C'[Run]$.

Thus $C[Run]$ simulates the behaviour of $C[P]$. The context C can be said to be *finitary* if the operational semantics of $C[Run]$ is finite state. It follows from the above enumeration that the operational semantics of $C[P]$ is finite state if C is finitary and P is finite state.

This is illustrated in Figure 1. Here $G[X] = a \rightarrow (X \;{}_{\{b,c\}}\|_{\{a,b\}}\; Q)$, with $Q = a \rightarrow b \rightarrow Q$, is shown on the process $P = b \rightarrow P'$, with $P' = c \rightarrow P'$.

The correspondence between the nodes of $G[P]$ and those of $G[Run]$ is shown directly. The tie-in between nodes of $G[P]$ and those of P is shown by the shading of the nodes (i.e. here, solid circle means P, empty means P').

If viewed in colour this figure illustrates the following pair of conventions:

- A node of $G[Run]$ is shown green if the argument process is active inside it, which it must be if at this point G can use one of its actions. A green node will allow a τ action of its argument in the simulation without

[3]This can be extended by letting P appear in multiple arguments of nondeterministic choice and guarded instances of $\Box$ (including multiple $Q(x)$ in the form $?x : A \rightarrow Q(x)$), because under these conditions P can never be alive twice in the same run.

[4]This argument applies to all finite behaviour models, and divergence-strict ones which have infinite behaviours. In other models infinite nondeterministic choice may not have the required properties.

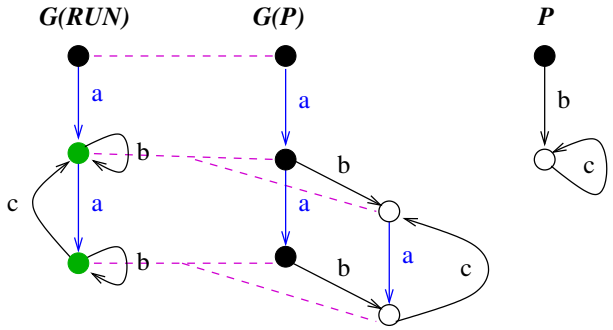

Figure 1: Simulating $G[P]$ from the graphs of $G[Run]$ and P.

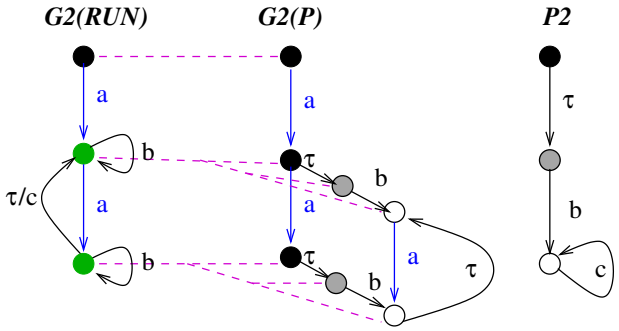

Figure 2: Example illustrating τ promotion and actions with different internal and external labels.

changing itself. In the picture the top node of $G[Run]$ is black, and the other two are green. Black nodes appear when either the argument is yet to be activated, or when it has been disposed of. If we were to replace P in the diagram by $P2 = (a \rightarrow P) \setminus \{a\}$, so it had the same LTS except there is an additional initial node with a τ to the old initial node, then the τ would be enabled (in the resulting $G[P2]$) in the two green states of G but not in the black one.

- Where a node of G can perform an action without the cooperation of its argument (i.e. when $G[STOP]$ has the action) the action is shown as blue. All actions from black states are blue, but they are also possible from green ones. In our example exactly the a events are blue: the first one because it is done by the prefix $a \rightarrow \cdot$ when the argument of G has not even been started, subsequent ones because they are performed by Q as part of the context.

Figure 2 illustrates the promotion of τ from green states as discussed above, and also the following additional convention:

- Each black (i.e. non-blue) action of $C[Run]$ both uses a named action of Run and generates a named action of $C[Run]$ (where the latter, but not

the former, name may be τ). If these names are not the same we just label the graph of $C[Run]$ by the two of them a/b, external name first.

In this second figure, $P2$ is as above and $G2(X) = a \to (X \;{}_{\{b,c\}}\|_{\{a,b\}}\; Q) \setminus \{c\}$, so each time the argument X is allowed to perform c, this is hidden and becomes τ.

With all the annotation we have described, it is possible in general to calculate the operational semantics of $C[P]$ for distributive contexts C from the graph of $C[Run]$ and the LTS of P.[5]

Our later analysis of the semantics of $C[P]$ relies heavily on this simulation and the correspondence it creates between the states of $C[P]$ and pairs (C', P') where $C'[Run]$ is a state of $C[Run]$ and P' is a state of P. That analysis, in considering the specification $Spec \sqsubseteq_X C[P]$, in effect unpicks the product illustrated in the two figures above in such a way as to factor $Spec$ by $C[\cdot]$.

2.2 Büchi automata

Büchi automata have been used in automated verification for decades [18]. They are classically used as the "liveness" half of splitting specifications into "safety" and "liveness" components, where a safety property is one whose failure can be detected in finite time, and a liveness one is a property only of infinite traces.

A *Büchi automaton* is a finite state automaton with labelled (by visible actions) transitions, an initial state s_0, with a subset of its states marked in some way, say starred. It accepts an infinite trace u just when it has an execution, starting from s_0, which performs u and passes infinitely often through one or more of these "accepting" starred states. While a finite automaton representing all its traces can only accept a closed set of infinite traces (if, for u, all finite prefixes s of u have an extension in S, then so is u), Büchi automata are more expressive and can represent more properties[6]. For example the automaton represented as the CSP process

$$\begin{array}{lcl} P & = & a \to P \;\Box\; b \to P \;\Box\; a \to S \\ S & = & a \to S \end{array}$$

with only S starred represents all the infinite traces of $\{a, b\}$ with only finitely many bs – which is not closed. One can couple this with the set of failures over $\{a, b\}$ where there is no refusal of the form $(s, \{a, b\})$ to characterise the most nondeterministic process that never deadlocks and where no infinite trace has infinitely many bs.

A Büchi automaton may or may not have a deterministic transition relation (i.e. have a unique state for every trace). Deterministic ones are strictly less expressive in the languages of infinite traces they accept: for example the automaton above is a standard nondeterministic[7] example where the language it represents cannot be expressed using a deterministic Büchi. Nevertheless the

[5] This view of their operational semantics is closely related to that given by the combinator style of semantics described in [13]: viewed from the combinator perspective, the syntactically enforced distributive restriction means that there is only at most one copy of the argument process present in each reachable combinator state.

[6] In fact they represent the ω-limits of regular languages, in common with several other formulations of automata.

[7] The transition a from P is nondeterministic: it can lead to P or S.

languages accepted by the deterministic case still need not be closed. For example consider the automaton with, in addition to s_0, one state $P(a)$ for each action $a \in \Sigma$. Every state accepts every $a \in \Sigma$, leading to the state $P(a)$. This is deterministic, and by starring just those $P(a)$ for a in some subset A of Σ, we characterise just those infinite traces that contain infinitely many actions from A.

Any Büchi (deterministic or nondeterministic) can be augmented to permit every finite trace without accepting any more infinite traces. This can be done by adding a single extra, unstarred state Ω which can take any action to itself and, from each state P of the original automaton, adding each action a not possible for that state to Ω. It is frequently convenient to assume they take this form. The other extreme, which usually adopt in automated analysis, is to remove all transitions and states after which no accepting state is reachable.

The languages represented by Büchi are closed under operations such as union, intersection and complementation. Indeed the standard way of verifying that a process P represented by an LTS satisfies the infinite trace property represented by a Büchi $\mathcal{B}$ is to discover if P has any trace in common with a Büchi representing the complement, or negation of $\mathcal{B}$. This fact is linked inseparably to the one that negating Büchi is a hugely complex and strongly exponential problem: see [17].

3 The expressibility of finite-state CSP refinement checking

In this paper we are interested in the practical question of what behavioural properties of CSP processes can be expressed via CSP for FDR. Since the processes allowed by FDR are finite-state, the most general form of check we can run on a process P is

$$F[P] \sqsubseteq G[P]$$

where $F[P]$ and $G[P]$ are both finitary CSP contexts, namely contexts which return a finite state process when given one. $\sqsubseteq$ means refinement over one of the CSP models.

In the absence of the finite-state restriction this problem was studied (in addition to non-behavioural properties) in [12], where it was proved for some models that behavioural properties were the same as distributive and refinement closed properties of the process P, and could all be expressed in the form $Spec' \sqsubseteq P$ Furthermore, it was shown that if G is any distributive context then $Spec \sqsubseteq G[P]$ is a behavioural property of P. Both of these proofs are trivial, but that of the first is entirely impractical because, as we demonstrated earlier, it builds $Spec'$ as the nondeterministic composition of all the processes satisfying the property.

In this paper we are concerned with what can be achieved using finite state $Spec$ and finitary contexts, so we will look more carefully at this case and obtain a similar result for these. In fact we will prove that for suitable finite behaviour models, finite state $Spec$ and finitary G, it is possible to create an equivalent refinement check $Spec' \sqsubseteq P$ where $Spec'$ is finite state. We will also derive a result for suitable divergence-strict models.

Non-distributive contexts do not naturally give rise to behavioural properties via $Spec \sqsubseteq C[P]$. This is explained in more depth in [12]. While it is possible to construct a context which is not syntactically distributive which is distributive over some or even all models, we doubt if doing so would broaden the range of behavioural properties that can be expressed. Because of this, and because of what we already demonstrated about the state space of $C[P]$ when $C[\cdot]$ is syntactically distributive, we restrict ourselves to syntactically distributive C when capturing the expressive power of CSP in creating behavioural specifications for FDR below.

We thus attempt to capture what properties are captured by checks of the form $Spec \sqsubseteq_X C[P]$ for such finitary contexts and finite-state *Spec*.

Theorem 1 *Let Spec be a finite-state process, and $G[\cdot]$ be a finitary distributive CSP context, then for each finite-behaviour CSP refinement $\sqsubseteq_X$, for X representing any of traces, failures, revivals or acceptance sets, the refinement $Spec \sqsubseteq_X G[P]$,* if satisfiable, *is equivalent (as a property of P) to a check $Spec' \sqsubseteq_X P$, where $Spec'$ is finite state.*

PROOFquad The analysis is a lot more difficult than the infinitary case mentioned above, but is based on examining how $G[\cdot]$ and P evolve together as illustrated in Figures 1 and 2. One can build the finite state $Spec'$ required in the next theorem by looking at the pairs (S', G') that can arise in the model checking of $Spec \sqsubseteq_X G[P]$, where S' is a normal form state of *Spec* and $G'[\cdot]$ is one of the "states" of $G[\cdot]$. The states of $Spec'$ are based on these, linked not by the actions that $G[P]$ performs between them, but rather the actions that P performs on the inside to enable the top-level ones. ∎

The full proof of this result can be found in the extended version of this paper. The authors conjecture that this result also holds for more complex forms of finite observation model such as refusal testing and acceptance traces (see [13]), but the proof, if it is to be on the same principles our one of the above result, would require a formidable amount of bookkeeping.

The form of this result is remarkably simple. We cannot achieve quite the same for models which place restrictions on divergence, because $G[P]$ might hide some actions of P with the same labels as other actions of P it does not hide. However the only way of getting a CSP model to recognise a bad infinite behaviour of P relative to a finite-state $Spec'$ is to have $G[P]$ diverge when the said infinite trace happens. We cannot create the necessary hiding of P by the form $Spec' \sqsubseteq_X P \setminus A$ for some A because of the ambiguity of hiding discussed above. In essence we need to ability to choose which events of P to hide irrespective of their names.

The way to do this is to use one-to-many renaming in a way similar to its use in demonstrating the expressiveness of CSP in [14]. As is usually the case for this construct we extend the alphabet Σ to $\Sigma_1 \cup \Sigma_2$ where $\Sigma_1 = \Sigma$ and Σ_2 is a second copy of each event in Σ. The copies of the event $a \in \Sigma$ are written a_1 and a_2 when the larger alphabet is in play. The renaming D maps each a to a_1 and a_2. We can then develop a *selective hiding context* (SHC) which leaves visible just those actions of P which, when P performs them in $C[P]$, guarantee something visible happens in $G[P]$. As Figure 1 shows, the occurrence of an action of P (there c) may guarantee that $G[P]$ does something visible (a) event though it is itself hidden.

Here a SHC takes the form

$$C[P] = (P[\![D]\!] \underset{\Sigma_1 \cup \Sigma_2}{\|} Reg) \setminus \Sigma_2$$

where *Reg* is a deterministic process with alphabet $\Sigma_1 \cup \Sigma_2$ which in every state accepts at least one of a_1 and a_2 for each $a \in \Sigma$. Thus $C[P]$ allows all the events of P and chooses which are hidden. In terms of the statement "$C[P]$ is divergence-free" the requirement that *Reg* is deterministic is unimportant, since replacing any finite-state process with the deterministic one with the same traces makes no difference to this check.

Theorem 2 *Let Spec be a finite-state process, $G[\cdot]$ be a finitary distributive CSP context, and X be one of the traces, failures, revivals or acceptance-set models. For each divergence-strict CSP refinement $\sqsubseteq_{XD}$, Spec $\sqsubseteq_{XD}$ $G[P]$ is equivalent (as a property of the divergence-free process P) to a pair of checks: Spec$' \sqsubseteq_X P$ (i.e. refinement over X with no divergence component) and $C[P]$ is divergence-free, for a SHC $C[\cdot]$.*

We have thus characterised the properties of infinite traces accessible through CSP in a concise form that we will use to relate to Büchi automata in the next section.

4 Divergence and deterministic Büchi automata

We will now demonstrate that the sets of infinite traces characterised by deterministic Büchi automata are identical to the sets such that the corresponding behavioural property of divergence-free P is decidable by the check $C[P]$ *is divergence-free* for some SHC C.

Showing that every deterministic Büchi $\mathcal{D}$ can be represented in our special CSP form is easy. Without loss of generality we can assume that $\mathcal{D}$ can perform every finite and infinite trace in Σ^*, without necessarily accepting all the infinite ones. The states of *Reg* are in $1-1$ correspondence with those of $\mathcal{D}$, with the addition of Run_{Σ_1} as above, and the actions of *Reg* are the same as those of $\mathcal{D}$ except:

- We now have two choices for which copy of each action in Σ to pick. Whenever an action is to an accepting state we choose the Σ_1 copy, which is not ultimately hidden, while each action leading to a non-accepting state of $\mathcal{D}$ becomes the Σ_2 copy in *Reg*.
- Where a state K of $\mathcal{D}$ has no event with a given label a, in *Reg* it has the action a_1 to Run_{Σ_1}.

Note that not only is *Reg* deterministic, but for each state K and each action $a \in \Sigma$, exactly one of a_1 and a_2 is an action of K, meaning it is a SHC regulator. Further, observe that $C[P]$ will diverge iff P performs an infinite trace on which $\mathcal{D}$ visits only finitely many accepting states. Hence, $C[P]$ is divergence free iff every infinite trace of P is accepted by $\mathcal{D}$, as required.

In the above we transformed a deterministic Büchi into a SHC by refraining from hiding those events that led to an accepting state. We can invert this process to turn a general SHC into a Büchi. However even though we can assume

that the regulator in the SHC is deterministic, we cannot assume that the Büchi is.[8] Whereas testing if the SHC corresponding to a deterministic Büchi $\mathcal{D}$ is divergence-free establishes if every infinite trace of the tested process is one of $\mathcal{D}$, because of the determinism it also tests if *any* of its infinite traces satisfy what we will refer to as co-$\mathcal{D}$, which accepts precisely those infinite traces with an execution passing through only finitely many of $\mathcal{D}$'s accepting states.

While the infinite traces accepted by $\mathcal{D}$ and co-$\mathcal{D}$ are disjoint, those accepted by $\mathcal{B}$ and co-$\mathcal{B}$ are not necessarily disjoint for a general Büchi $\mathcal{B}$, because a given infinite trace may have two executions in $\mathcal{B}$, one of which passes through infinitely many accepting states and the other does not. For a general SHC $C[\cdot]$, corresponding to Büchi $\mathcal{B}$, $C[P]$ being divergence-free for divergence-free P means that P has no infinite trace satisfying co-$\mathcal{B}$, or in other words that all P's infinite traces lie in the complement of those of co-$\mathcal{B}$, which is a possibly proper subset of the infinite traces satisfying $\mathcal{B}$, on the assumption that $\mathcal{B}$ can perform any infinite trace at all.

In the literature, a co-Büchi automaton is, like co-$\mathcal{B}$, a finite state automaton with a set of marked states, whose accepting condition on infinite traces is that there is an execution of it that only passes through finitely many marked states. In [9] Miyano and Hayashi show that moving from deterministic to general co-Büchi automata does not change the languages of infinite traces that can be characterised. This is exactly what we need, since it shows that the traces of P that do not cause a general SHC to diverge are those of some deterministic Büchi, namely one whose complement is exactly the same as co-$\mathcal{B}$, where $\mathcal{B}$ is the general Büchi corresponding to the SHC.

We can thus conclude the following result.

Theorem 3 *The specifications of divergence-free processes P that are captured by Spec $\sqsubseteq_{XD}$ $G[P]$ for G a distributive CSP context are precisely the same as the combinations*

$$Spec' \sqsubseteq_X P \qquad \text{and} \qquad P \text{ satisfies } \mathcal{D}$$

for Spec′ a finite state process and $\mathcal{D}$ a deterministic Büchi.

Furthermore, we note that the construction above which showed how satisfaction of a deterministic Büchi can easily be transformed into the divergence-freedom of a SHC applied to P, also works for a general Büchi $\mathcal{P}$ in the sense that divergence freedom implies that all P's infinite traces are in the complement of those accepted by co-$\mathcal{B}$. Thus it is perhaps best to think of CSP's divergence checks as demonstrating the infinite traces lie in the complement of a co-Büchi.

5 Combining FDR and Büchi automata

While the above characterisation is satisfying, it shows that the properties representable by nondeterministic Büchi cannot all be verified using existing FDR functionality, at least using the form of check we have been examining. In this section we show how FDR can be extended to do this. Whereas the analysis

[8] It is deterministic when the regulator has exactly one of a_1 and a_2 in each state, rather than both being allowed.

above carefully examined and used the expressive power of CSP, the solution here is largely independent of the CSP notation and closely follows what other tools have done to solve this problem.

In general, we want to be able to check if all of the infinite traces of a process are contained within the infinite traces of a Büchi automaton. We have extended FDR to check this by verifying, like other tools, that no infinite trace of the process is accepted by the negation of the Büchi automata. Since, as discussed earlier, negating a Büchi automata is impractical, we instead require the user to give the negated property.

As input, FDR takes a file that describes the processes to verify, along with a list of assertions to verify about the processes. These assertions can check simple properties such as deadlock or divergence freedom and can, more generally, check for refinement in the three standard CSP denotational models. We have extended the range of assertions to include an *impossible for* assertion that checks for Büchi acceptance. For example, the following fragment uses FDR's new functionality to check if a process P can perform only finitely many bs:

```
P = ...

B = buchi(FiniteBs, accept)
FiniteBs = a -> FiniteBs [] b -> FiniteBs [] a -> FiniteBs'
FiniteBs' = accept -> FiniteBs' [] a -> FiniteBs

assert B :[impossible for]: P
```

The function `buchi` takes a process P and an event *accept*, and constructs a Büchi automaton that has the infinite traces of P, but where any state that can perform *accept* is marked as accepting. For example, the Büchi automaton for `B` will consist of two states, one for `FiniteBs` and one for `FiniteBs'` where the latter is marked as accepting, as it can perform the `accept` event (this arc is also removed from the Büchi automaton). This function can be applied to arbitrary CSP processes, which makes it possible to define Büchi automata using parallel compositions, or other complex CSP constructs.

The assertion `assert B :[impossible for]: P` passes precisely when no infinite trace of `P` is also accepted by the Büchi automata `B` (i.e. all infinite traces of `B` are *impossible for* `P`). FDR checks the impossible for assertions using the standard technique of a nested depth-first search [**?**] on the product of the property (which is given by the user in the negated form) and the system. When this fails, the algorithm produces a counter-example, consisting of a path through the product automata from the starting state to a cycle containing an accepting state. FDR displays this counter-example to a user in a natural way. Further, FDR's existing compositional debug facilities can be used to *divide* the counterexample to work out how each component of the system contributes to the erroneous acceptance.

As this implementation is only a prototype, we have not performed any performance analysis. However, there is no reason to suppose that a FDR-based version of this algorithm would perform differently from any of the other implementations of nested depth-first search in other tools, such as SPIN and LTSmin. If this extension proves valuable, we intend to develop a high-performance version.

It would have been possible to check the impossible for assertions using existing functionality of FDR, as per [6]. However, this requires putting the product automata on the left-hand side of a refinement check. This is undesirable for two reasons. Firstly, this would mean that FDR could not compute a counter-example when the assertion failed, making this feature difficult to use. Secondly, the left-hand side of a refinement check has to be normalised which is PSPACE-hard (although it is often less in practice). Together, this makes such an approach wholly impractical. In contrast, the nested depth-first search approach takes linear time, and returns counterexamples when the assertion fails. It is also worth noting that efficient multi-core algorithms are known for checking for the existence of an accepting cycle [**?**, 8, **?**], whilst normalisation is likely to be challenging to parallelise.

6 Conclusions and future work

We have considerably clarified what sort of behavioural specifications are possible using the conventional operation of FDR, and shown how it extends to a natural characterisation of a behavioural specification as the combination of a process representing the finitely observable behaviour and a Büchi for the infinite trace specification.

It is interesting to reflect on what this characterisation says about CSP. CSP's one-to-many renaming operator is crucial in defining SHC's and in generating contexts where divergence freedom naturally reduces to nondeterministic co-Büchis rather than deterministic ones. On the other hand the use of double renaming that we made in showing that the complement of any co-Büchi can be specified by a CSP divergence freedom check is *not* essential. Whereas we used this type of renaming to distinguish between actions to accepting seats (visible) and ones to other states (invisible), the same effect as a specification would have been achieved by getting each visit to an accepting state to inject an extra visible action, with all ordinary actions now being hidden. This latter construction shows that CSP's separation between hiding and parallel is not essential to express these divergence-based conditions: in fact it would equally work with a combination of CCS parallel and restriction.

While the conventional interpretation of CSP with environments controlling all visible events does not fit easily with fairness, there are circumstances where fairness is naturally desirable, such as in conjunction with the shared variable front end SVA [5, 13] which is capable of analysing programs written in a simple imperative language with shared variables. The new support for Büchi analysis allows us to perform verifications under fairness assumptions. We intend to report separately on the use of our extended FDR functionality to support this.

It would, in principle, be easy to implement general LTL specifications efficiently for FDR, extending the respectively inefficient and partial embeddings described in [6, 7].

We intend to explore the potential for extending CSP's refinement-based compositional verification to support specifications that are the conjunction of a specification of the form $Spec \sqsubseteq_X P$ and a Büchi.

The current implementation of nested DFS within FDR is merely a prototype that allows us to experiment with Büchi automata. If the implementation proves useful, we will implement a high-performance multi-core variant of either nested

depth-first search [**?**], or of Tarjan's algorithm [8, **?**].

The extensions to FDR outlined in this paper (namely the `B :[impossible for]: P` assertion) are not yet available in the main release of FDR. Interested parties should contact one of the authors for access to a version that includes this extension.

Acknowledgements We are grateful to Michael Goldsmith for suggesting the `impossible for` terminology. Research into FDR3 has been partially sponsored by DARPA under agreement number FA8750-12-2-0247.

References

[1] Stephen D Brookes. A model for communicating sequential processes. 1983.

[2] Stephen D Brookes, A.W. Roscoe, and David J Walker. An operational semantics for CSP. *Oxford University Technical Report*, 1986.

[3] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A.W. Roscoe. FDR3: a modern refinement checker for CSP. In *Proceedings of TACAS*, LNCS 8413, 2014.

[4] C.A.R. Hoare. *Communicating sequential processes*. Prentice Hall, 1985.

[5] David Hopkins and AW Roscoe. SVA, a tool for analysing shared-variable programs. 2007.

[6] Michael Leuschel, Andrew Currie, and Thierry Massart. How to make fdr spin LTL model checking of CSP by refinement. In *FME 2001: Formal Methods for Increasing Software Productivity*, pages 99–118. Springer, 2001.

[7] Gavin Lowe. Specification of communicating processes: temporal logic versus refusals-based refinement. *Formal Aspects of Computing*, 20(3):277–294, 2008.

[8] Gavin Lowe. Concurrent depth-first search algorithms based on tarjans algorithm. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2015.

[9] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32(3):321–330, 1984.

[10] A.W. Roscoe. Model-checking CSP. In *A classical mind: Essays in honour of C.A.R. Hoare*. 1994.

[11] A.W. Roscoe. *The theory and practice of concurrency*. Prentice Hall, 1997.

[12] A.W. Roscoe. On the expressive power of CSP refinement. *Formal Aspects of Computing*, 17(2), 2005.

[13] A.W. Roscoe. *Understanding concurrent systems*. Springer, 2010.

[14] A.W. Roscoe. On the expressiveness of CSP. 2011.

[15] S.A. Schneider. *Concurrent and real-time systems*. Wiley New York, 2000.

[16] Fabio Somenzi and Roderick Bloem. Efficient büchi automata from ltl formulae. In *Computer Aided Verification*, pages 248–263. Springer, 2000.

[17] Moshe Y Vardi. Büchi complementation: A forty-year saga. In *5th symposium on Atomic Level Characterizations (ALC05)*, 2005.

[18] Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society, 1986.

A Proofs of Theorems 1 and 2

Proof of Theorem 1 We deal first with the case of the finite traces model and then indicate how to extend this proof to the other models. Consider model checking $Spec \sqsubseteq_T G[Run]$ in FDR's usual way except that we look for the full set of reachable state pairs (namely normal form states S' of *Spec* and states $G'[Run]$ of the RHS such that the truth of the assertion requires $S' \sqsubseteq G'[Run]$ to be true), carrying on enumerating even after one or more errors have been found (i.e. cases (S', G') where $G'[Run]$ can perform an event not permitted by S'). This creates a new LTS which we will call R.

Note that R may contain both visible (Σ) and τ actions, but that if there is a τ from (S', G') to (S'', G'') then $S' = S''$.

We add an extra node E (for error) to R and add a "red" action labelled a to E from each node (S', G') where $G'[Run]$ has an action a not permitted by S'.

We now colour the nodes of R. For such a node $N = (S', G')$ (representing the state-pair $(S', G'[Run])$) it may be the case that $G'[STOP]$ has an action illegal for S'. In that case we will label N red.

Wherever an edge in R corresponds to a blue action (as defined earlier) of the graph of $G[Run]$, we colour it blue in R. We can extend the red colouring to any node from which an existing red node is reachable along blue edges. Let the black nodes be all nodes not now coloured red.

Now for each black node (S', G') we form the set $Blue(S', G')$ of R-nodes consisting of (S', G') itself and all other nodes reachable along blue edges from it. The intuition behind $Blue(S', G')$ is that if, in exploring the refinement of $Spec \sqsubseteq G[P]$, we reach the state pair $(S', G'[P'])$, then each of the state pairs $(S'', G''[P'])$ for $(S'', G'') \in Blue(S', G')$ is also reachable. It follows that if any of these P'' make $G''[P'']$ perform an action forbidden for S'', then the overall refinement fails.

We form a new graph W with the black nodes of R plus E, whose actions are now based not on those of $G[Run]$ but instead on those performed by the underlying Run when the given transition occurs. Because Run performs no action on blue transitions in R, there is no action in W paralleling a blue action directly, but if from (S', G') one can reach (S'', G'') by passing through any number (zero or more) blue actions in R to (S^*, G^*) and then through a single black action to (S'', G'') that resulted from Run in $G' * [Run]$ performing a, then (S', G') has the action a in W to (S'', G''). We can call these W's black actions. Where in R, the state (S', G') has an action x to either E or a red state in which $G'[Run]$ performs a, or alternatively a series of blue actions to a state (S^*, G^*) where similar x caused by a, then (S', G') has the action a in W to E.

The intuition here is that if, in the checking of $(S, G[P])$, we ever arrive at a state $(S', G'[P'])$, with P's own actions taking us through the graph W to (S', G'), then if P' can perform any action a of a red edge, then the refinement check is doomed to failure because it either has (or can via blue edges) reached a point from which an action is possible which denies the check.

We can now form $Spec'$ as follows: first we normalise the process represented by the root node of W after adding a new action *error* from E to itself to ensure that E can clearly be distinguished by its behaviour from the behaviour of every other node. This normal form NW has a unique result state for each trace s

of *Run* that the exploration of $Spec \sqsubseteq_T G[Run]$ enables up to and including the first event outside the traces of *Spec*. We can tell by the fact that this state – always a set of nodes of W – contains E that *Run* or any other process performing this trace can lead $G[P]$ outside the traces of *Spec*.

Some of these sets X may contain E. Prune away all the actions of such states aside from the *error* action to themselves. We do this because we want to ignore any differences in the behaviour of nodes that only become apparent after the failure of the refinement check is inevitable.

We then re-normalise to get NNW, where now the only state with the *error* event is the one (which can again call E) with only this action to itself.

We form $Spec'$ with one node for each node of NNW other than E. (If the root node of NW contains E then the specification $Spec \sqsubseteq G[P]$ is unsatisfiable; even $P = STOP$ fails it.) A given node X of NNW will, for each action $a \in \Sigma$, have one of the following

- The action a to a node of NNW other than E
- The action a to E.
- No action a at all.

This last possibility occurs when no state that $G[Run]$ can reach in the refinement check as *Run* performs a trace s that leads through NNW to X ever permits its argument to perform a. This will typically happen because of parallel composition in $G[\cdot]$.

The state of $Spec'$ corresponding to X has each action of the first type leading to the node corresponding to node in NNW that a leads to. It has no actions with the second class of labels. It has each action of the third type leading to the process *Run*.

The interpretations of these three actions (which in the refinement check $Spec \sqsubseteq_T G[P]$ would be performed by P) are as follows:

- The first is an action that some reachable $G'[\cdot]$ does let the current state of P perform, but which does not lead inevitably to the check failing. If P does then perform such an action, we need to track the refinement check forward.
- The second is the same, only the refinement check does now inevitably fail. So we want the refinement $Spec' \sqsubseteq_T P$ to fail when P performs this action. Denying $Spec'$ this action (in the corresponding state) has exactly this effect.
- The final one is an action that no G' will allow to happen, so it does not matter if the present state of P can do it, and this leads to no condition on whatever state P itself, reaches after the action.

We can thus conclude that $Spec' \sqsubseteq_T P$ if and only if $Spec \sqsubseteq_T G[P]$, which is what we were seeking to prove.

In the above proof, while we considered the possibility that $G[P]$ performs actions x that are τs either independently of P or because (as when $G[P]$ involves hiding) of visible actions of P, we have not considered the τ actions of $G[P]$ which arise whenever its argument is active and itself performs a τ. There is in fact no need to consider them for finite trace checks since such τs never

change the value of either S' or G' and of course do not themselves make the refinement fail since they are not counted in the trace.

We now turn to the question of other finite behaviour models of CSP. For many purposes[9] the finite behaviour models fall into two categories: namely the ones where any structure other than finite traces is only found at the end of traces (traces, failures, revivals, acceptance sets) and those where there are observations along traces as well (refusal testing and acceptance traces). The normal forms of the first group retain the property of having a single state per traces, whereas the second group have more complex structures.

This fact means that the proof of the rest our theorem, which involves only the first sort of model, is a relatively straightforward elaboration on the trace on. Again we establish the obligations on each state of P by tracking which combined states (S', G') of a normal form state of *Spec* and a context state G' can arise for them. The states S' now carry obligations on what can be (for example) refused on them, as well as what events can happen next. Where $G'[Run]$'s initial events include τs, a state $G'[P]$ may or may not be stable depending on what events P offers, with $G'[P]$ only exhibiting any refusal if P refuses all the events that $G'[P]$ turns into an initial τ. Just as, in the traces case, G' might make a pair (S', G') unsatisfiable because of the events it can do by itself, similarly G' might single-handedly make S' impossible to satisfy in one of the more elaborate models. Because **div** (the divergent process) is the most refined element of each of these models (and equivalent to $STOP$ over $\mathcal{T}$), this happens whenever $S' \not\sqsubseteq_X G'[\mathbf{div}]$; in this case (S', G') becomes a red state.

This completes our proof. The authors conjecture that this result also holds for more complex forms of model such as refusal testing, but the proof, if it is to be on the same principles as above, would require a formidable amount of bookkeeping.

We could have eliminated the states of *Spec* from the above proof by rolling them up into G: it is known [**?**] that any finitary check $Spec \sqsubseteq_T P$ is equivalent to one of the form $STOP \sqsubseteq (W_{Spec} \underset{\Sigma\setminus\{bark\}}{\|} P) \setminus (\Sigma \setminus \{bark\})$ for a finite-state *watchdog* process based on *Spec* which communicates the new event *bark* at any point where P's trace as become illegal for *Spec*.

It follows that $Spec \sqsubseteq_T G[P]$ is equivalent to $STOP \sqsubseteq (W_{Spec} \underset{\Sigma\setminus\{bark\}}{\|} G[P]) \setminus (\Sigma \setminus \{bark\})$, so taking

$$G^+(P) = (W_{Spec} \underset{\Sigma\setminus\{bark\}}{\|} G[P]) \setminus (\Sigma \setminus \{bark\})$$

would have meant we could have developed the construction of $Spec'$ above without needing to keep track of S'. However the actual $Spec'$ that is constructed would be essentially identical to the one of our earlier construction.

Proof of Theorem 2 Without loss of generality we can assume the following of $G[\cdot]$: if Q_s is the process which simply performs the finite trace s and does not diverge, then $G[Q_s]$ is divergence-free for all $s \in \Sigma^*$. If our original $G[Q_s]$ can diverge, it means the $G[P]$ can diverge with P only performing a finite trace. In other words $G[\cdot]$ has some state G' such that $G'[STOP]$ diverges.

We can get exactly the same specification of P by replacing G by one which replaces the state G' by $STOP$, and having *Spec* additionally forbid those traces

[9] For example the applicability of the FDR *diamond* compression.

(necessarily those of a finite state process) where $G[Q_s]$ diverges but *Spec* does not. In other words we have identified an additional set of finite traces of P which $Spec \sqsubseteq_{XD} G[P]$ forbids due to the divergence aspect of this check, and moved them over to "where they belong" in the finitary part. Henceforth we will assume that no state of $G[\cdot]$ has such a divergence through blue τs, using the description from the previous proof.

The benefit of this assumption is that now $G[P]$ can diverge only when $G[\cdot]$ takes P through an infinite trace of actions, all but finitely many of which are directly or indirectly hidden, becoming τs of $G[P]$. Thus $G[P]$ being divergence free says exactly that P has none of the infinite traces that allow this, and is thus precisely a specification on the infinite traces of P.

For $Spec \sqsubseteq_{XD} G[P]$ we require the following

- On traces s where *Spec* does not diverge we require that $G[P]$ does not diverge.
- On traces s where *Spec* does not diverge we require that all of $G[P]$'s finitely observable behaviour (next actions, and where appropriate refusals etc) is permissible for *Spec* over X.
- On traces s where *Spec* can diverge, no constraint whatsoever is placed on $G[P]$.

It follows that $Spec \sqsubseteq_{XD} G[P]$ holds if and only if the following pair of assertions hold:

- $G[P] \underset{\Sigma}{\|} R$ is divergence-free, where R a the finite state process with precisely the non-divergent traces of *Spec*. (For example R can be obtained by removing all divergent states from the normal form of *Spec*.)
- $Spec \sqsubseteq_X G^*(P)$, where G^* is $G[P] \underset{\Sigma}{\|} R'$, where R' is the *pseudo-deterministic* process with the same traces and divergences as *Spec*. In other words, its traces are the those of *Spec* that do not have a proper divergent prefix, after a divergent trace all it can do is perform an infinity of τs, and after a non-divergent trace t it can accept (and never refuse) each of the next events of $Spec/t$. The parallel composition with R' has the effect of removing any information about what $G^*(P)$ after the point where *Spec* diverges.

Therefore the theorem is proved if we can establish that a general finitary distributive context $G[P]$ can be reduced to the specific form of context allowed in the statement of the theorem with respect to the test "is divergence free".

The idea behind the structure:

$$(P[\![D]\!] \underset{\Sigma_1 \cup \Sigma_2}{\|} Reg) \setminus \Sigma_2$$

is that *Reg* only allows P to go down paths that it can within $G[P]$, and that each occurrence of a P event that is hidden within $G[P]$ is turned into a member of Σ_2, while any occurrence of a P event that becomes a visible event of $G[P]$ is turned into a member of Σ_1, remaining visible in our fixed style of context.

Note that if we can find a finite-state *Reg* satisfying what is required, we can also find a deterministic one: the deterministic process with the same traces,

noting that a finite-state process always has the same infinite traces as the deterministic one thus constructed.

We will therefore not bother to make *Reg* deterministic in what follows. In fact we will give it one state for each state of $G[\cdot]$. The state corresponding to a given $G'[\cdot]$ will allow P to perform exactly the actions that $G'[\cdot]$ and all $G''[\cdot]$ reachable from it along blue actions do. By "allow P to perform" we mean that *Reg* can perform the said action a in P's alphabet, except when the $G''[P]$ converts an action of P into a τ and every blue action leading to that point[10] is also τ, then *Reg* picks the Σ_2 rather than Σ_1 copy of a. Thus a given state may have either, both or neither of the copies of a given a. The result(s) of each action in the resulting *Reg* are those corresponding to the state(s) reachable after zero or more blue actions and the one generated by the P action in $G[Run]$. It should be clear that this achieves exactly what is required.

[10]Figure 2 illustrates the fact than an infinite behaviour of $G2(P2)$ need not be a divergence even though every action in which $P2$ participates is hidden. Here each hidden c is preceded by an unhidden blue a.