Resource Based Models for Asynchrony*

J. Rathke
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova
via Dodecaneso 35, 16146 Genova, Italy
julianr@cogs.susx.ac.uk

Abstract

We propose a new graph-based approach to modelling asynchronous languages and show how the new model can be viewed as a collapse of the standard transition system model for asynchronous behaviour by utilising the commuting properties of asynchronous transitions.

The motivation behind these new models stems from the issue of regularity for asynchronous processes. We note that the class of regular processes fails to contain many useful asynchronous processes and we identify a larger subclass of BPP accordingly. We call this new class *asynchronously regular processes.*

Using the new models we provide two appealing abstract characterisations of asynchronous bisimulation equivalence, namely, as spans of open maps and as a winning strategies for a bisimulation game. Also, by exploiting the coincidence of finite graphs with regular processes we see that bisimulation is polynomial time decidable over our class of asynchronously regular processes.

1 Introduction

It is becoming increasingly clear that the nature of output messages in languages such as the asynchronous π-calculus, [2, 6], Pict [13] and the Join-calculus, [4] is one of persistent resources. Recently, this persistence of output was exposed at the level of transition systems by identifying certain commuting properties guaranteed of asynchronous systems [14]. Given such a situation, it would seem reasonable to question whether transition systems afford a good representation of asynchronous processes. After all, the ordering of transitions in a graph is used merely to reflect the precedence of actions which can be performed by the process. The distinguishing feature of output actions is that they cannot preclude other actions; so why model them as transitions?

Our approach is to view output messages purely in terms of resources. Our models, resource graphs, have no output transitions but instead record the availability of output resources as computation progresses. This might be achieved by allowing each node to be a pair containing some 'state' of the system along with the multiset of resources which are currently available. In fact, we see in Section 3.1 that this is pretty much how the transition system model behaves so little is to be gained from this solution. A much more compact representation is possible if we don't explicitly record the current resources available but simply see how resources *become* available. We augment each input and τ transition with the multiset of outputs which become available as a result of performing this transition. It should be clear that we will also need to store the information of which resources are initially available in a system. For example, the process

$$P = c! \parallel a?(b! \parallel b! \parallel Q) + \tau.(d! \parallel R)$$

has an initial resource $\{c!\}$ and two immediate transitions $P \xrightarrow{a?}$ and $P \xrightarrow{\tau}$ which release the resources $\{b!, b!\}$ and $\{d!\}$ respectively. We represent these two transitions as

$$P \overset{a,\{b,b\}}{\rightsquigarrow} Q \text{ and } P \overset{\tau,\{d\}}{\rightsquigarrow} R,$$

where the input/output sense of actions is now implicit. This move to recording resources on edges rather than at nodes allows many more *infinite state* processes to be modelled by finite resource graphs.

*On leave from the University of Sussex. Supported by the EU-HCM Express network.

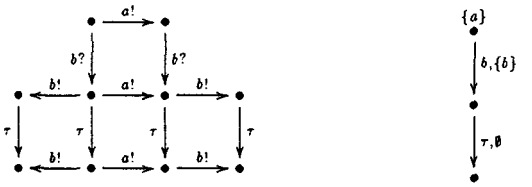


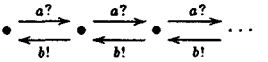
Figure 1: Transition system and resource graph for $a! \parallel (b?(b! \parallel \tau.\ \mathbf{nil}))$

To contrast the standard transition system models with the resource graph interpretation of a process consider the example process in Figure 1. The redundancy in the transition system model is highlighted well by the uniform shape of asynchronous transition systems imposed by Selinger's axioms [14]. We know, owing to the asynchronous nature of the language, that the $a!$ is possible at the initial node and, until it is used, will continue to be available, thus in the resource graph model this information is utilised to create a more compact graph.

The models for the process

$$P = a?(b! \parallel P)$$

are more illuminating. This process will in fact be modelled by an infinite transition system,



yet the structure of the process is very simple — at all times there is an $a?$ action possible and for each $a?$ action performed an additional $b!$ resource becomes available. Initially there are no $b!$ resources available. In fact, this gives us a resource graph with a single node, initial resource set is empty and there is a single looping transition

So far we have shown how we could tailor transition systems to be more suited to modelling asynchronous processes. But we must consider how this would actually benefit us. The examples show us that we immediately have a more compact representation of systems, so this could clearly be useful when it comes to checking equivalence of processes. Ideally we could check bisimulation between processes by building their resource graphs and checking some kind of bisimulation on these. This would necessitate defining the appropriate notion of bisimulation for resource graphs. Given such a situation, we would easily obtain a decision procedure for checking bisimilarity for the class of processes which receive finite resource graph models.

It is well known that finite state transition systems correspond (up to strong bisimulation) to *regular* processes in CCS, that is processes which make no use of the static operators, parallel composition and restriction underneath recursion [9, 10]. If we forbid the use of parallel composition and restriction under recursion from asynchronous CCS we lose a great deal of expressive power, in fact, we lose the ability to perform more than a finite number of output actions. This sorry state of affairs would mean that even the paradigmatic asynchronous buffer process

$$\mathbf{rec}\ X.a?(a! \parallel X)$$

is not expressible. This restricted use of parallelism is certainly too strong for asynchronous languages and we must consider a weaker notion of regularity. We propose that a parallel composition $p \parallel q$ in the scope of recursion binders be allowed providing that either p or q is merely an output message – we call such processes asynchronously regular. The class of asynchronously regular processes would now include the asynchronous buffer process shown above as well as many other *infinite state* processes. Moreover, all such processes will be modelled by finite resource graphs.

In order to define bisimulation on resource graphs we appeal to the abstract definition proposed by Joyal, Nielsen, Winskel [7]. This definition simply requires us to choose a suitable category in which to observe basic computation paths. Using intuition gleaned from [1] to choose our notion of morphism of resource graphs, we see that the notion of asynchronous bisimulation proposed by [1] exists in an

abstract form. The key to our choice of morphism lies in understanding internal τ actions as a pair of unspecified synchronising actions, hidden from the environment. One may like to think of τ prefixing as syntactic sugar for

$$\nu a.(a! \parallel a?P).$$

We consider what effects specifying a name, a, for these actions, and allowing them to be seen by the environment, has; call this specified synchronising pair τ_a, so one might think of τ_a prefixing as

$$a! \parallel a?P.$$

To define our notion of morphism on resource graphs we discuss general considerations about morphisms of labelled graphs. We think of a morphism

$$f : G \longrightarrow G'$$

between two labelled graphs as representing that G' is a refinement of G. That is to say that G is more specified than G'. A morphism should refine transitions of the graph in some way. We will outline what we understand by refinement.

Transitions represent both local communication, τ moves, and capacity for interacting in a more global sense, $a?$ and $a!$ moves. Given a process p, we can observe the global computations it can engage in by inducing them using an environment process e situated in parallel with p. We say that e *offers* an action $a!$, say, if $e \xrightarrow{a!}$, and that p *accepts* this offer if it synchronises with e to perform an internal reduction or computation. A transition $p \xrightarrow{\alpha} p'$ of some transition system can be understood then as saying that the least offer one need make p to observe some synchronisation and reduction to p' is $\hat{\alpha}$, where $\hat{a}! = a?$, $\hat{a}? = a!$, and $\hat{\tau}$, specified or not, is empty. The ordering on offers is simply that the empty offer is less than all other offers, which are incomparable. We will expect that any τ_a transition can be refined by a τ transition because we have information about the computation yielded by τ_a, the name of the channel on which synchronisation occurs, that we do not have of the computation given by the τ action. We say that a computation is *covert* if we do not know the name of the synchronising channel. All computations induced by τ prefixes are covert. Using this definition we say that

$$p \xrightarrow{\alpha} p' \ f\text{-refines} \ q \xrightarrow{\beta} q'$$

if p maps to q and q maps to q' under the morphism f, such that the least offer $\hat{\alpha}$ made to p can also be accepted by q to induce a reduction to q'. If the induced computation of p is covert then the corresponding induced computation of q must also be covert. More precisely we ask that

$$\hat{\alpha} \parallel p \longrightarrow p' \text{ implies } \hat{\alpha} \parallel q \longrightarrow q'$$

such that if we don't know the name on which p synchronises then we cannot know the name on which q synchronises. We can see that following refinements hold for transition systems,

$$\begin{array}{lll} p \xrightarrow{\alpha} p' & f\text{-refines} & f(p) \xrightarrow{\alpha} f(p') \\ p \xrightarrow{\tau_a} p' & f\text{-refines} & f(p) \xrightarrow{\tau} f(p'), \end{array}$$

and these give rise to a fairly unsurprising definition [16] of morphism for asynchronous transition systems. However, we observe a peculiarity in the category of resource graphs. Edges of resource graphs are labelled with pairs, $m \overset{\alpha,S}{\rightsquigarrow} m'$. Refinement of these edges will have to take into account the resources which are collected. To spell this out we say

$$m \overset{\alpha,S}{\rightsquigarrow} m' \ f\text{-refines} \ n \overset{\beta,S'}{\rightsquigarrow} n'$$

if m maps to m', and n maps to n' such that the least offer $\hat{\alpha}$ which (covertly) reduces m to state m' with S extra resources can also be accepted by n so that the (covert) reduction induced takes us to state n' with the same extra resources. Under this definition we have the following refinements

$$\begin{array}{lll} m \overset{\alpha,S}{\rightsquigarrow} m' & f\text{-refines} & f(m) \overset{\alpha,S}{\rightsquigarrow} f(m') \\ m \overset{a,S+\{a\}}{\rightsquigarrow} m' & f\text{-refines} & f(m) \overset{\tau_a,S}{\rightsquigarrow} f(m') \\ m \overset{\tau_a,S}{\rightsquigarrow} m & f\text{-refines} & f(m) \overset{\tau,S}{\rightsquigarrow} f(m'). \end{array}$$

The second refinement holds because the least offer $a!$ made to m can be accepted by $f(m)$ to reduce to $f(m')$ with S extra resources, along with the extra a resource which was unused by the τ_a.

By considering refinement to be transitive we can dispense with the idea of $m \overset{\tau_a, S}{\rightsquigarrow}$ transitions for resource graphs altogether and simply use $m \overset{a, S+\{a\}}{\rightsquigarrow} m'$ instead. The chief feature of our resource graphs morphisms then is that a morphism from R to R' allows us to specify in R, a name for an internal synchronisation in R'. We reinforce these intuitions by exploiting the game theoretic characterisation of bisimulation to highlight the rôle of τ synchronisations as specified and unspecified pairs of actions.

We briefly outline the structure of the remainder. The following short section recalls the category of transition systems and describes the asynchrony axioms. In Section 3 we define our category of resource graphs and relate them to transition systems. Bisimulation equivalence is defined as the span of open maps in this category and we characterise it using bisimulation like relations. The game theoretic description of this equivalence is spelled out in Section 4. We demonstrate the usefulness of our models in Section 5 by giving an enhanced notion of regularity for asynchronous systems and prove that bisimulation equivalence is polynomial time decidable over this class. Section 6 contains our conclusions.

Acknowledgments: The author(s) would like to thank Catuscia Palamidessi and Guy McCusker for carefully reading a draft of this paper and suggesting many improvements. Thanks also to Colin Stirling for providing some useful pointers in the literature. This work was carried out during research visits at INRIA, Sophia-Antipolis and the University of Genova, which were funded by the EU-HCM Express network. I would sincerely like to thank Catuscia Palamidessi and Ilaria Castellani and their respective institutions for their extreme generosity and for directing me in my research.

2 Asynchronous systems

We recall, from [16], the definition of the category of transition systems, $\mathcal{TS}$ and describe the sub-category, $\mathcal{ATS}$, of asynchronous transition systems, as characterised by Selinger.

Firstly, objects of $\mathcal{TS}$ are transition systems, $(\mathcal{N}, n_0, L, \longrightarrow)$ where n_0 is a specified initial node. Morphisms in $\mathcal{TS}$ are pairs of morphisms

$$(\sigma, \lambda) : (\mathcal{N}, n_0, L, \longrightarrow) \to (\mathcal{N}', n_0', L', \longrightarrow)$$

such that $\sigma : \mathcal{N} \to \mathcal{N}'$ and $\lambda : L \rightharpoonup L'$ is a partial function with the property that

$$n \xrightarrow{\alpha} n' \text{ implies } \begin{cases} \sigma n \xrightarrow{\lambda\alpha} \sigma n' & \text{if } \lambda\alpha \downarrow \\ \sigma n = \sigma n' & \text{otherwise.} \end{cases}$$

Composition of morphisms is given by pairwise (partial) function composition and the identity morphisms are simply pairs of identity functions on the respective sets.

A morphism $(\sigma, \lambda) : T \to T'$ indicates that T' is a refinement of T in the sense that T is more specified than T'. Observe that T may have more atomic actions than T' with extra transitions pertaining to these actions. Also, T may have a more specific structure than T', with less non-determinism and fewer transitions. Indeed, when the λ component of a morphism is the identity then this morphism is simply an inclusion of T in T'. This idea of a morphism being a refinement is examined again in the category of resource graphs.

The particular sub-category $\mathcal{ATS}$ of $\mathcal{TS}$ in which we are interested is described as follows. Objects of the category are transition systems whose label set L is typed, that is

$$L \subseteq \mathcal{A} \times \{!, ?, \tau\} \cup \{\tau\}$$

where $\mathcal{A}$ is some set of channel names. That is, each action is either an output, $a!$, an input $a?$, the result of a synchronisation of these τ_a, or an internal, hidden, synchronisation, τ. These transition systems are subject to certain axioms, presented in Figure 2 which characterise their asynchronous behaviour [14].

Morphisms of $\mathcal{ATS}$ are similar to morphisms in $\mathcal{TS}$ except that the relabelling component λ is now a partial function on $\mathcal{A}$. We write τ_a^o to mean either τ_a or τ and define $\lambda a! = (\lambda a)!$, $\lambda a? = (\lambda a)?$, $\lambda\tau = \tau$, and $\lambda\tau_a = \tau^o_{\lambda a}$. Composition and identities are defined as in $\mathcal{TS}$.

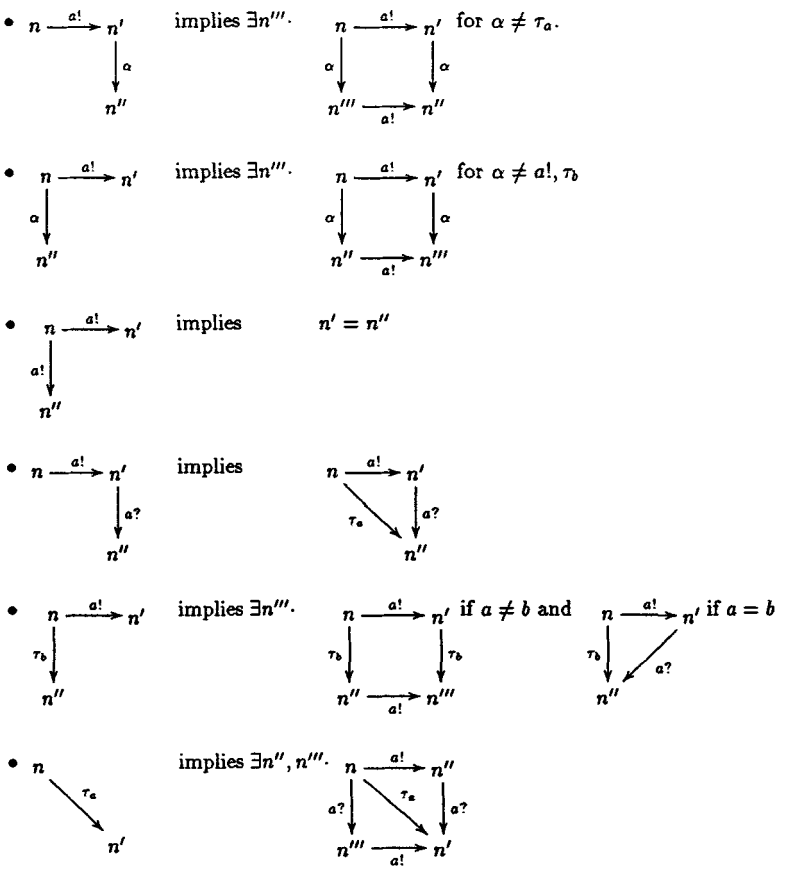


Figure 2: Axioms for asynchronous transition systems

3 Resource graphs

A *resource graph* is a graph based model for systems in which there is some notion of resource, that is, some action which is persistent and not subject to reactive behaviour. A resource's use is never precluded. The particular application we have in mind is for modelling asynchronous systems wherein the ! actions of the systems are considered as resources.

Formally, a resource graph is a quintuple

$$(\mathcal{M}, \mathcal{A}, m_0, S_0, \rightsquigarrow)$$

where $\mathcal{M}$ is a set of nodes, $\mathcal{A}$ is some set of names, m_0 is a specified initial node in $\mathcal{M}$ and S_0 is a multiset of resources which are initially available. We write $\mathcal{A}^{**}$ for the set of all multisets over the set $\mathcal{A}$. So we see that $S_0 \in \mathcal{A}^{**}$. The edges of the graph are given by

$$\rightsquigarrow \subseteq \mathcal{M} \times (\mathcal{A} \cup \{\tau\}) \times \mathcal{A}^{**} \times \mathcal{M}$$

and we write $m \overset{a,S}{\rightsquigarrow} m'$ if $(m, a, S, m') \in \rightsquigarrow$. We will use $+$ and $-$ to denote union and difference where multiset difference $S - S'$ is a partial operator and is only defined when $S' \subseteq S$. These operators are extended pointwise to multiset valued functions.

We can now describe our category of resource graphs, in fact we describe two. The first, $\mathcal{RG}$, has morphisms similar to the category $\mathcal{ATS}$ in that a morphism represents refinement by introducing extra atomic actions and embedding. We use this category to relate the standard transition system models to resource graph models. The second category we define, $\mathcal{RGA}$, contains morphisms which, following the ideas outlined in the introduction, also allow refinement by specifying on which name a synchronisation takes place. The two categories are such that $\mathcal{RG}$ is a lluf sub-category of $\mathcal{RGA}$.

The objects of $\mathcal{RG}$ are resource graphs. A morphism $(\sigma, \lambda, \varphi)$ from

$$R = (\mathcal{M}, \mathcal{A}, m_0, S_0, \rightsquigarrow)$$

to

$$R' = (\mathcal{M}', \mathcal{A}', m_0', S_0', \rightsquigarrow)$$

is a triple where σ is a function $\mathcal{M} \to \mathcal{M}'$, λ is a partial function $\mathcal{A} \cup \{\tau\} \to \mathcal{A}' \cup \{\tau\}$ which preserves τ and φ is a function $\mathcal{M} \to \mathcal{A}'^{**}$ such that the following conditions are satisfied:

(i) $\sigma m_0 = m_0'$

(ii) $\lambda S_0 + \varphi m_0 = S_0'$

(iii) $m \overset{\alpha,S}{\rightsquigarrow} m'$ implies

$$\begin{cases} \sigma m \overset{\lambda\alpha,S'}{\rightsquigarrow} \sigma m' \text{ where } S' = \lambda S + \varphi m' - \varphi m & \text{if } \lambda\alpha \downarrow \\ \sigma m = \sigma m' \text{ and } \lambda S = \emptyset, \varphi m = \varphi m' & \text{otherwise.} \end{cases}$$

The φ component of a morphism allows for a resource graph to be embedded within a larger resource graph containing additional resources available at each node. Identity morphisms in $\mathcal{RG}$ are of the form $(Id, Id, C_\emptyset)$ where $C_\emptyset$ denotes the constant empty multiset function. Composition is defined by

$$(\sigma, \lambda, \varphi); (\sigma', \lambda', \varphi') = (\sigma; \sigma', \lambda; \lambda', (\varphi; \lambda' + \sigma; \varphi')).$$

It is straightforward enough to check that $\mathcal{RG}$ is indeed a category.

3.1 Relating the transition system and resource graph models

We describe an adjunction $\mathcal{ATS} \underset{ra}{\overset{ar}{\underset{\dashv}{\rightleftarrows}}} \mathcal{RG}$ between our category of asynchronous transition systems and our simple category of resource graphs. The counit of this adjunction is in fact an isomorphism so the adjunction is a reflection.

The functor $ra : \mathcal{RG} \to \mathcal{ATS}$ acts on objects as follows:

$$ra(\mathcal{M}, \mathcal{A}, m_0, S_0, \rightsquigarrow) = (\mathcal{M} \times \mathcal{A}^{**}, \mathcal{A}, (m_0, S_0), \longrightarrow)$$

where $\longrightarrow$ is defined by

$$\begin{array}{lll} (m, S + \{a\}) \xrightarrow{a!} (m, S) & & \\ (m, S) \xrightarrow{a?} (m', S') & \text{if } m \overset{a,S''}{\rightsquigarrow} m' & \text{and } S' = S + S'' \\ (m, S) \xrightarrow{\tau} (m', S') & \text{if } m \overset{\tau,S''}{\rightsquigarrow} m' & \text{and } S' = S + S'' \\ (m, S + \{a\}) \xrightarrow{\tau_a} (m', S') & \text{if } m \overset{a,S''}{\rightsquigarrow} m' & \text{and } S' = S + S''. \end{array}$$

On morphisms we have that $ra(\sigma, \lambda, \varphi) = (\sigma', \lambda)$ where $\sigma'(m, S) = (\sigma m, \lambda S + \varphi m)$.

In the other direction we need a couple of preliminary definitions before we can describe ar. Firstly, given an asynchronous transition system we let $\asymp$ denote the least equivalence on its nodes such that

$$n \xrightarrow{a!} n' \text{ implies } n \asymp n'.$$

Secondly, we write $n \xrightarrow{S!}$ if there exists a (possibly infinite) sequence of transitions

$$n \xrightarrow{a_1!} n_1 \xrightarrow{a_2!} \ldots \xrightarrow{a_k!} n_k \xrightarrow{a_{k+1}!} \ldots$$

such that $\sum_k a_k = S$. Define $Outs(n)$ to be the maximum S such that $n \xrightarrow{S!}$.

We can now describe our functor ar. On objects:

$$ar(\mathcal{N}, n_0, L, \longrightarrow) = (\mathcal{N}/\asymp, \mathcal{A}, [n_0], Outs(n_0), \rightsquigarrow)$$

where $\mathcal{A}$ is the first projection of the label set $L \subseteq \mathcal{A} \times \{!, ?, \tau\}$ and $\rightsquigarrow$ is defined by

$$\begin{array}{l} [n] \overset{a,S}{\rightsquigarrow} [n'] \text{ if } n \xrightarrow{a?} n' \\ [n] \overset{\tau,S}{\rightsquigarrow} [n'] \text{ if } n \xrightarrow{\tau} n' \end{array} \quad \text{and } S = Outs(n') - Outs(n).$$

The reader is invited to check that the asynchrony axioms guarantee that $Outs(n) \subseteq Outs(n')$, thus ensuring that this does define a resource graph.

On morphisms we have that $ar(\sigma, \lambda) = ([\sigma], \lambda, (\sigma; Outs(\_) - Outs(\_); \lambda))$ where $[\sigma][n] = [\sigma n]$ and the third component is applied to any representative of the $\asymp$ equivalence class. This is a well-defined resource graph morphism because of the asynchrony axioms.

Theorem 3.1 *ar is left adjoint to ra, moreover the counit, $ra; ar \xrightarrow{\varepsilon} Id$, of the adjunction is an isomorphism.*

Proof: The counit of the adjunction is $(\varepsilon, Id, C_\emptyset)$ where $\varepsilon([(m, S)]) = m$. This is easily seen to be natural and universal and it has an inverse $(Id, \varepsilon^{-1}, C_\emptyset)$ where $\varepsilon^{-1}(m) = [(m, \emptyset)]$. Dually, the unit of the adjunction is $(Id, [\_] \times Outs(\_))$. □

We see that the unit of the adjunction does not necessarily have an inverse. This is because in mapping our resource graph to a transition system we consider all configurations of nodes and multisets. This includes many configurations which don't necessarily arise during computation. Thus, if we restrict our attention to those configurations which are *reachable*, in some sense, then we can find an inverse for our unit.

To this end, define the set of *reachable configurations* of a resource graph to be $Reach(m_0, S_0)$ where $Reach$ is defined inductively as follows:

$$\begin{array}{lcl} Reach_0(m, S) & = & \emptyset \\ Reach_{n+1}(m, S) & = & \{(m, S') \mid S' \subseteq S\} \cup \bigcup_{m \overset{a,S''}{\rightsquigarrow} m'} Reach_n(m', S'' + S). \end{array}$$

Let $Reach(m, S) = \bigcup_{n \geq 0} Reach_n(m, S)$.

We immediately note that all reachable configurations of the resource graph $ar(T)$ are of the form $([n], Outs(n))$ for some $n \in T$. Thus, by replacing the set of all configurations $\mathcal{M} \times \mathcal{A}^{**}$ by just the reachable ones, $Reach(m_0, S_0)$, we can obtain an equivalence between the sub-categories of $\mathcal{ATS}$ and $\mathcal{RG}$ whose graphs only contain reachable states.

3.2 A larger category of resource graphs

We now consider a slightly more general category $\mathcal{RGA}$ of which $\mathcal{RG}$ is a lluf sub-category, that is, the objects of $\mathcal{RGA}$ are exactly the objects of $\mathcal{RG}$. The extension lies in the notion of morphism. We relax the definition of morphism of resource graphs in accordance with the motivation outlined in the introduction. The generalisation is tantamount to allowing a τ action of the target graph to be specified as a synchronisation on a particular name. We argued that a synchronisation on channel a is a refinement of the action $a?$ where an extra $a!$ resource is made available. The new notion of morphism utilises this observation.

A morphism of $\mathcal{RGA}$ is a triple $(\sigma, \lambda, \varphi)$ as above, however we ask that the following conditions be satisfied instead:

(i) $\sigma m_0 = m_0'$ as above

(ii) $\lambda S_0 + \varphi m_0 = S_0'$ as above

(iii) $m \overset{\tau,S}{\rightsquigarrow} m'$ implies $\sigma m \overset{\tau,S'}{\rightsquigarrow} \sigma m'$

(iv) $m \overset{a,S}{\rightsquigarrow} m'$ implies

$$\begin{cases} \sigma m \overset{\lambda a,S'}{\rightsquigarrow} \sigma m' \text{ or} & \text{if } \lambda a \downarrow \\ \sigma m \overset{\tau,S''}{\rightsquigarrow} \sigma m' & \\ \sigma m = \sigma m' \text{ and } \lambda S = \emptyset, \varphi m = \varphi m' & \text{otherwise} \end{cases}$$

where $S' = \lambda S + \varphi m' - \varphi m$ and $S'' = (\lambda(S - \{a\})) + \varphi m' - \varphi m$. Identities and composition are defined as in $\mathcal{RG}$ and $\mathcal{RGA}$ is also seen to be a category.

3.3 Bisimulation on resource graphs

We propose a definition of bisimulation, suitable for resource graphs, in abstract form. Namely, we use the machinery of open maps, [7], to declare two resource graphs with label the same label set $\mathcal{A}$, bisimilar if there exists a span of open maps between them in the sub-category $\mathcal{RGA}_o$ of $\mathcal{RGA}$. All of this sub-category's objects have label set $\mathcal{A}$ and all morphisms have the identity as the λ component. Furthermore, edges in the graphs of $\mathcal{RGA}_o$ enjoy the following determinacy conditions:

$$\begin{array}{llll} m \overset{a,S}{\rightsquigarrow} m' & \text{and} \quad m \overset{a,S'}{\rightsquigarrow} m' & \text{implies} & S = S' \\ m \overset{a,S+\{a\}}{\rightsquigarrow} m' & \text{and} \quad m \overset{\tau,S'}{\rightsquigarrow} m' & \text{implies} & S = S' \end{array}$$

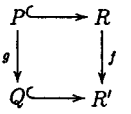
One should note that this determinacy condition is a technical restriction and can easily be enforced in an arbitrary resource graph by simply sending offending pairs of transitions to different targets.

We define *paths* in $\mathcal{RGA}_o$ to be resource graphs of the form

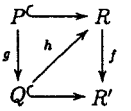
$$m_0 \overset{a_1,S_1}{\rightsquigarrow} m_1 \overset{a_2,S_2}{\rightsquigarrow} \cdots \overset{a_k,S_k}{\rightsquigarrow} m_k$$

with initial node m_0 and initial resources S_0.

Recall that we call a morphism $f : R \to R'$ *open* if for all paths P, Q such that the following commutes

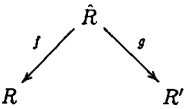


then we have a morphism $h : Q \to R$ such that



(we use $\hookrightarrow$ to denote inclusion morphisms).

Define bisimulation then as $R \sim_o R'$ iff there exists a



with f, g open. It is easy to see that $\sim_o$ is both reflexive and symmetric, but to prove that it is transitive it is sufficient to check that $\mathcal{RGA}_o$ has pullbacks [7].

Proposition 3.2 *$\mathcal{RGA}_o$ has pullbacks, which makes $\sim_o$ an equivalence relation.*

3.4 Characterising $\sim_o$

The abstract definition of bisimulation using open maps, while being quite general, is not particularly illuminating. For this reason it is natural to seek simpler characterisations of this relation.

To this end we consider the following class of relations. For resource graphs

$$(\mathcal{M}, \mathcal{A}, m_0, S_0, \rightsquigarrow) \text{ and } (\mathcal{M}', \mathcal{A}, m_0', S_0', \rightsquigarrow)$$

such that $S_0 = S_0'$ we call a symmetric relation $\mathcal{B}$ on $\mathcal{M} \times \mathcal{M}'$ a *resource graph bisimulation* if $(m_0, m_0') \in \mathcal{B}$ and whenever $(m_1, m_2) \in \mathcal{B}$ then

- if $m_1 \overset{\tau, S}{\rightsquigarrow} m_1'$ then there exists a m_2' such that $m_2 \overset{\tau, S}{\rightsquigarrow} m_2'$ with $(m_1', m_2') \in \mathcal{B}$
- if $m_1 \overset{a, S}{\rightsquigarrow} m_1'$ then there exists a m_2' such that $m_2 \overset{a, S}{\rightsquigarrow} m_2'$ or $m_2 \overset{\tau, S'}{\rightsquigarrow} m_2'$ with $(m_1', m_2') \in \mathcal{B}$, and $S' + \{a\} = S$.

We write $R \sim_{rg} R'$ if there exists a resource graph bisimulation relating R and R'.

Theorem 3.3 *$\sim_{rg}$ and $\sim_o$ coincide.*

3.5 A model for asynchronous CCS

We recall the notion of asynchronous bisimulation, $\sim_{as}$, as proposed by Amadio, Castellani, Sangiorgi [1] (albeit for the π-calculus and without τ_a actions) and show that the functor ar and the equivalence $\sim_o$ provide a fully abstract interpretation for $\sim_{as}$.

A symmetric relation $\mathcal{B}$ on asynchronous CCS processes is called an asynchronous bisimulation if whenever $(p, q) \in \mathcal{B}$ we have

- if $p \xrightarrow{a!} p'$ then there exists a q' such that $q \xrightarrow{a!} q'$ with $(p', q') \in \mathcal{B}$.
- if $p \xrightarrow{\tau_a} p'$ then there exists a q' such that $q \xrightarrow{\tau_a^o} q'$ with $(p', q') \in \mathcal{B}$.
- if $p \xrightarrow{\tau} p'$ then there exists a q' such that $q \xrightarrow{\tau} q'$ with $(p', q') \in \mathcal{B}$.
- if $p \xrightarrow{a?} p'$ then there exists a q' such that $q \xrightarrow{a?} q'$ with $(p', q') \in \mathcal{B}$ or $q \xrightarrow{\tau_a^o} q'$ with $(p', a! \parallel q') \in \mathcal{B}$.

Recall that τ_a^o means either τ_a or τ. The largest such relation will be denoted $\sim_{as}$.

By considering asynchronous processes as asynchronous transition systems, via operational semantics, we can interpret processes as resource graphs by means of the functor ar. This interpretation is fully abstract for $\sim_{as}$.

Theorem 3.4 *For asynchronous processes p and q, $p \sim_{as} q$ if and only if $ar(p) \sim_o ar(q)$.*

Proof: Show $p \sim_{as} q$ iff $ar(p) \sim_{rg} ar(q)$ and use Theorem 3.3. □

The reader should note that $\sim_{as}$ is an atypical notion of bisimulation for transitions systems and differs from the one in [1] in that τ actions must be matched solely by τ actions, thereby disallowing the possibility of matching with a τ_a action. A more standard notion of equivalence is gained by replacing the third matching condition above with

$$\text{if } p \xrightarrow{\tau} p' \text{ then there exists a } q' \text{ such that } q \xrightarrow{\tau_a^o} q' \text{ with } (p', q') \in \mathcal{B}.$$

Let $\sim_{as}^+$ denote the equivalence yielded by this modification. This situation is of course rather unsatisfactory in general, but we can at least console ourselves with the fact that $\sim_{as}$ coincides with the more standard $\sim_{as}^+$ on the class of transition systems for which *Outs* is always finite at each node. In particular $\sim_{as}$ and $\sim_{as}^+$ coincide on our class of regular processes in Section 5.

Proposition 3.5 *$\sim_{as} q$ and $\sim_{as}^+$ coincide on the class of transition systems such that Outs is finite at each node.*

Proof: One inclusion is immediate. For the reverse inclusion we need to show that $\sim^+_{as}$ is an asynchronous bisimulation. The only way that $\sim^+_{as}$ may fail to be an asynchronous bisimulation is if, given $p \sim^+_{as} q$ we have $p \xrightarrow{\tau} p'$ being matched by $q \xrightarrow{\tau_a} q'$ for some q'. We show that there must be a matching τ transition in this case. Now, we know that $Outs(p)$ is finite and that each of these output transitions from p must be matched by q. Therefore there exist p_0, q_0 such that

$$p \xrightarrow{a_1!} \cdots \xrightarrow{a_n!} p_0 \quad \text{and} \quad q \xrightarrow{a_1!} \cdots \xrightarrow{a_n!} q_0,$$

$Outs(p_0) = Outs(q_0) = \emptyset$ and $p_0 \sim^+_{as} q_0$. We know that asynchrony ensures $p_0 \xrightarrow{\tau} p'_0$ for some p'_0 and that this must be matched by $q_0 \xrightarrow{\tau} q'_0$ because q_0 can no longer perform a τ_a transition as $Outs(q_0) = \emptyset$. Again, by asynchrony we know that $q \xrightarrow{\tau} q''$ for some q''. It is easy to check that $p' \sim^+_{as} q''$ follows from $p'_0 \sim^+_{as} q'_0$. □

4 Game theoretic description of $\sim_o$

We extend our characterisation of asynchronous bisimulation further by showing how the notion can be captured as winning strategies of a suitable game. The use of games to characterise bisimulation has provided a conceptually powerful tool for understanding bisimulation as an equivalence which captures interaction [15]. In our setting the game characterisation helps us understand the rôle of τ as a pair of unspecified, complementary actions.

We give a general definition of what we mean by a *game* and instantiate this definition later to give us our appropriate equivalence. So, a game Γ, is a quadruple $(\mathcal{C}, c_0, \rhd, \lambda)$ where $\mathcal{C}$ is a set of configurations with a specified initial configuration c_0. The relation $\rhd \subseteq \mathcal{C} \times \mathcal{C}$ comprises the rules of the game. This relation tells us how play may continue from one move to the next. The function $\lambda : \mathcal{C} \to \{O, P\}$ labels moves as either Opponent or Player moves according to who is next to play - we require $\lambda c_0 = O$ and $\lambda c \neq \lambda c'$ whenever $c \rhd c'$. A play of a game is a sequence

$$c_0 \rhd c_1 \rhd c_2 \rhd \cdots \rhd c_k \rhd \cdots$$

We write $P(\Gamma)$ for the set of all plays and abuse notation by writing λcs to mean the label of the last move of cs (if it exists). A play, cs, is called maximal if it is infinite or cannot be extended, that is there is no move c such that $cs \rhd c$.

We say that O wins the finite play cs if $\lambda cs = P$ and cs is maximal. Dually, we say that P wins a (possibly) infinite play if $\lambda cs = O$ and the play is maximal. A strategy for O is a partial function from $Pos(O) = \{cs \mid \lambda cs = O\}$ to $M(P) = \{c \mid \lambda c = P\}$. We can define a strategy for P similarly.

Given an O-strategy π_o, we write $P(\pi_o)$ for

$$\{cs \in P(\Gamma) \mid \forall cs' \sqsubset cs \cdot \lambda cs' = O \text{ implies } (cs' \rhd \pi_o(cs')) \sqsubset cs\}$$

where $\sqsubset$ is the prefix ordering on plays. We say that the strategy π_o is winning if all maximal plays of $P(\pi_o)$ are finite and labelled P.

Dually, we can define $P(\pi_p)$ for player strategies π_p and say that π_p is winning if all maximal plays of $P(\pi_p)$ are infinite or labelled O.

4.1 The asynchronous bisimulation game

We can now describe the game which characterises asynchronous bisimulation simply by describing the configurations of the game and the rules. Before formally defining these however, we give an intuitive explanation of the game.

Imagine a table containing a pile of cards, labelled with names from some set $\mathcal{A}$, arranged in such a way as to model a resource graph. In addition to this pile of cards there is a hand of cards kept as a reserve. So, if the resource graph has a $m \overset{a,S}{\rightsquigarrow} m'$ transition, this means there will be an a card available for play from the pile. If it is played then the cards in S must be picked up and kept in the reserve hand and the pile of cards will now reflect state m'. If the resource graph has a $m \overset{\tau,S}{\rightsquigarrow} m'$ transition then the player has a blank card available. If she wishes to play this blank card she must pencil in a name, play

Left Rules: If $d \in \{L, E\}$

Table:

$$((m,S),(m',S'),zs,d) \rhd ((m'',S+S''),(m',S'),a?zs,\bar{d})$$
$$\text{if } m \overset{a,S''}{\rightsquigarrow} m'' \text{ and } d = L \text{ implies } hd(zs) = a?$$

Reserve:

$$((m,S),(m',S'),zs,d) \rhd ((m,S-\{a\}),(m',S'),a!zs,\bar{d})$$
$$\text{if } d = L \text{ implies } hd(zs) = a!$$

Blank:

$$((m,S),(m',S'),zs,d) \rhd ((m'',S+S''+\{a\}),(m',S'),a?zs,\bar{d})$$
$$\text{if } m \overset{\tau,S''}{\rightsquigarrow} m'' \text{ and } d = L \text{ implies } hd(zs) = a!$$

Right Rules: If $d \in \{R, E\}$

Table:

$$((m,S),(m',S'),zs,d) \rhd ((m,S),(m'',S'+S''),a?zs,\underline{d})$$
$$\text{if } m' \overset{a,S''}{\rightsquigarrow} m'' \text{ and } d = R \text{ implies } hd(zs) = a?$$

Reserve:

$$((m,S),(m',S'),zs,d) \rhd ((m,S),(m',S'-\{a\}),a!zs,\underline{d})$$
$$\text{if } d = R \text{ implies } hd(zs) = a!$$

Blank:

$$((m,S),(m',S'),zs,d) \rhd ((m,S),(m'',S'+S''+\{a\}),a?zs,\underline{d})$$
$$\text{if } m' \overset{\tau,S''}{\rightsquigarrow} m'' \text{ and } d = R \text{ implies } hd(zs) = a!$$

where $\bar{L} = E$, $\bar{E} = R$ and $\underline{R} = E$, $\underline{E} = L$.

Figure 3: Rules for asynchronous bisimulation game

it, pick up the cards in S for the reserve hand and in addition to these must fill in a blank card with the same name and place it in the reserve hand. A card from the reserve hand may be played irrespective of the pile of cards representing the resource graph.

A configuration of our game is a pair of the above tables, that is, two tables with a pile of cards and a separate reserve hand each. At each turn, Opponent can play a card from either table and Player must play the same card from the other table. The only extra condition is that a card from a reserve hand is played by Player if and only if Opponent has played her card from a reserve hand.

Opponent always starts and play continues until one of the players becomes stuck. Opponent wins if Player becomes stuck and Player wins otherwise.

To formalise this, given two resource graphs

$$R = (\mathcal{M}, \mathcal{A}, m_0, S_0, \rightsquigarrow) \text{ and } R' = (\mathcal{M}', \mathcal{A}, m_0', S_0', \rightsquigarrow)$$

we describe the game $\Gamma_{\mathcal{A}}(R, R')$ as the quadruple $(\mathcal{C}, c_0, \rhd, \lambda)$ where $\mathcal{C}$ is the set of all

$$((m,S),(m',S'),zs,d)$$

such that $m \in \mathcal{M}$, $m' \in \mathcal{M}'$, $S, S' \in \mathcal{A}^{**}$, $zs \in (\mathcal{A} \times \{!,?\})^{**}$ and $d \in \{L, R, E\}$. Clearly, the nodes of the resource graphs represents the pile of cards on the tables and the respective multisets represent the reserve hands. We use the list zs to represent the cards that have already been played and d merely to indicate which table must be played from next, the *Left*, *Right* or Either. The cards in zs are tagged with a ! or a ? to indicate whether the card was played from a table or a reserve hand. It should be no surprise then that the initial configuration is

$$c_0 = ((m_0, S_0), (m_0', S_0'), \varepsilon, E).$$

We can label moves by using the last component so that $\lambda c = P$ if $d \in \{L, R\}$ and $\lambda c = O$ if $d = E$. The rules for the game are given in Figure 3 and fall into three pairs of symmetric rules which describe the moves of playing a card from the table, the reserve hand and playing a blank card by penciling in a name.

We write $R \sim_\Gamma R'$ if there exists a winning Player strategy according to the rules of $\Gamma_A(R, R')$. It is simple enough to see that this is indeed an equivalence relation, in fact this is exactly resource graph bisimulation.

Theorem 4.1 *$\sim_{rg}$ coincides with $\sim_\Gamma$.*

Proof: It is easy to see that $\sim_{rg} \subseteq \sim_\Gamma$. For the reverse inclusion, given a winning strategy, it is sufficient to build a bisimulation relation. This is constructed as pairs of nodes which occur in the configurations of plays according to the winning strategy. We take exactly those pairs which occur after Player moves. To see that this will be a resource graph bisimulation we note that τ transitions must be matched by τ transitions — otherwise Opponent could win by choosing a fresh name to pencil in on the blank card given by the τ action. Player couldn't hope to match this unless he had also had a τ move available. To see that the resources being collected by each graph must be identical we note that, otherwise, Opponent could win by simply playing a move from the larger of the two reserve hands. □

5 Regular asynchronous processes

We hinted earlier that our new model would lend itself to providing a notion of *regular* process for asynchronous calculi whereby regular terms have finite graphs. By *finite* graph we mean finitely many nodes, finitely many transitions and each resource multiset is finite. So far we have interpreted asynchronous CCS in $\mathcal{RG}$ indirectly by first giving an $\mathcal{ATS}$ semantics and then applying the functor ar. This approach suffices for modelling our language; indeed, to establish a regular term/finite resource graph relationship one need only show that the equivalence relation used by the functor ar has finite index on transition systems generated by regular terms. However, this method is slightly unsatisfactory as it involves building potentially infinite graphs and collapsing them. What would be more pleasing is a direct interpretation of $aCCS$ in $\mathcal{RGA}$ by which regular terms immediately receive finite graph models. Furthermore, we should require that this interpretation be compositional and coincides (up to equivalence) with the indirect interpretation.

In fact, for our purposes it suffices to interpret what we will refer to as *(asynchronously) regular* terms of $aCCS$. These can be characterised by the following grammar

$$p ::= \mathbf{nil} \mid X \mid a! \parallel p \mid p \parallel a! \mid \sum_I \alpha_i.p_i \mid \mathbf{rec}\, X.p$$

where I is a finite indexing set, X is drawn from some set of variables Var, the α_i are either $a?$ or τ and all recursions are guarded. We adopt the conventional notions of free and bound variables here.

To interpret recursion, we take the approach of [9] and augment resource graphs with an extra component. This new component, $\lhd$ is a relation on nodes of the graph and the ambient set of recursion variables, Var. We say that a variable, X, is *unguarded* at a node m if $m \lhd X$ and we call a resource graph *closed* if $\lhd$ is the empty relation.

We make use of the following operators on resource graphs: firstly, we note that resource graphs have a tensor product structure, $\otimes$, with unit I. Given graphs

$$R = (\mathcal{M}, \mathcal{A}, m_0, S_0, \rightsquigarrow, \lhd)$$

and

$$R' = (\mathcal{M}', \mathcal{A}', m_0', S_0', \rightsquigarrow, \lhd')$$

this is defined in the obvious way as

$$R \otimes R' = (\mathcal{M} \times \mathcal{M}', \mathcal{A} + \mathcal{A}', (m_0, m_0'), S_0 + S_0', \rightsquigarrow_\otimes, \lhd \cup \lhd')$$

where

$$(m,n) \overset{\alpha,S}{\rightsquigarrow}_{\otimes} (m',n) \quad \text{if } m \overset{\alpha,S}{\rightsquigarrow} m'$$
$$(m,n) \overset{\alpha,S}{\rightsquigarrow}_{\otimes} (m,n') \quad \text{if } n \overset{\alpha,S}{\rightsquigarrow} n'.$$

The tensor unit is $I = (\{\bullet\}, \emptyset, \bullet, \emptyset, \emptyset, \emptyset)$. The definition of $\otimes$ easily lifts to morphisms to become a bifunctor on $\mathcal{RGA}$.

We interpret an output action $a!$ as the resource graph

$$(\{\bullet\}, \{a\}, \bullet, \{a\}, \emptyset, \emptyset)$$

and we will refer to this graph simply by $a!$. Similarly, use the name X to refer to the resource graph

$$(\{\bullet\}, \emptyset, \bullet, \emptyset, \emptyset, \{(\bullet, X)\}).$$

Another useful operation is that of the lifted sum of resource graphs. Given an I indexed set of graphs R_i, an I indexed set of actions α_i, and a multiset S, we define

$$\sum(\alpha_i, R_i) = ((\bigcup \mathcal{M}_i) + \{\bullet\}, \bigcup \mathcal{A}_i \cup \{\alpha_i \mid \alpha_i \neq \tau\}, \bullet, \emptyset, \rightsquigarrow_{\perp}, \bigcup \lhd_i)$$

where $\rightsquigarrow_{\perp} = \bigcup_i \rightsquigarrow_i \cup \left\{\bullet \overset{\alpha_i,S_{0_i}}{\rightsquigarrow} m_{0_i}\right\}$.

Finally, we describe how we interpret recursion over resource graphs. Given a graph R, we define $\mathbf{rec}\ X.R$ to be the graph

$$(\mathcal{M}, \mathcal{A}, m_0, S_0, \rightsquigarrow_+, \lhd_+)$$

where $\lhd_+$ is just $\lhd$ with all pairs (m, X) removed. $\rightsquigarrow_+$ is defined in two steps. Firstly, define

$$m \overset{\alpha,S}{\rightsquigarrow}_1 m' \quad \text{if} \quad m \overset{\alpha,S}{\rightsquigarrow} m' \quad \text{and } m' \not\lhd X$$
$$m \overset{\alpha,S+S_0}{\rightsquigarrow}_1 m' \quad \text{if} \quad m \overset{\alpha,S}{\rightsquigarrow} m' \quad \text{and } m' \lhd X.$$

Then, let $m \overset{\alpha,S}{\rightsquigarrow}_+ m'$ if $m_0 \overset{\alpha,S}{\rightsquigarrow}_1 m'$ and $m \lhd X$, or $m \overset{\alpha,S}{\rightsquigarrow}_1 m'$.

The informed reader will notice that this definition of recursion differs slightly from that in [9] and is not sufficient to model general recursion, but we exploit the property that regular terms never have more than one unguarded variable to give a simple definition.

These operators now allow us to interpret regular terms of $aCCS$ in the desired manner:

$$\begin{array}{lcl}
[\![\ \mathbf{nil}\ ]\!] & = & I \\
[\![X]\!] & = & X \\
[\![a! \parallel p]\!] & = & a! \otimes [\![p]\!] \\
[\![p \parallel a!]\!] & = & [\![p]\!] \otimes a! \\
[\![\sum \alpha_i.p_i]\!] & = & \sum(\alpha_i, [\![p_i]\!]) \\
[\![\mathbf{rec}\ X.p]\!] & = & \mathbf{rec}\ X.[\![p]\!].
\end{array}$$

Let $\hat{p}$ denote the transition system that would model p using the standard SOS semantics of CCS.

Proposition 5.1

(i) The resource graph $[\![p]\!]$ is finite for any regular term p.

(ii) If p is closed then $[\![p]\!]$ is a closed graph.

(iii) Every finite closed graph is $\sim_{rg}$ equivalent to $[\![p]\!]$ for some regular p.

(iv) $ar(\hat{p}) \sim_{rg} [\![p]\!]$.

This firmly establishes the correspondence between asynchronously regular terms and finite resource graphs.

5.1 Deciding bisimulation equivalence

To see the usefulness of having finite models we need only look at the problem of deciding bisimulation equivalence. It is evident that $\sim_{as}$ will be a decidable equivalence over asynchronously regular terms due to work on infinite state transition systems [3]. Specifically, asynchronously regular terms are a small subclass of BPP and bisimulation equivalence is decidable over this class of processes. What is not clear however is the complexity of this decision procedure. The proofs that bisimulation equivalence is decidable over BPP do not provide any upper bounds for the decision procedure [5, 11]. The class of asynchronously regular processes are much simpler than BPP and therefore allow us to find such bounds. In fact, because our models for this class are finite then standard techniques apply [8, 12].

Theorem 5.2 *Asynchronous bisimulation equivalence, $\sim_{as}$, is decidable in polynomial time for (asynchronously) regular processes.*

Proof: In order to decide $P \sim_{as} Q$, by Proposition 5.1, Proposition 3.5 and Theorem 3.4 it is sufficient to check $[\![P]\!] \sim_{rg} [\![Q]\!]$. We know by Proposition 5.1, (i) that these resource graphs are finite. The decision procedure now follows by first checking the initial resource sets of each graphs, and then solving the partition refinement problem of [12] for the finite set of relations

$$\begin{aligned} mE_{\tau,S}m' \quad &\text{if} \quad m \overset{\tau,S}{\rightsquigarrow} m' \\ mE_{a,S}m' \quad &\text{if} \quad m \overset{a,S}{\rightsquigarrow} m' \\ mE^{+}_{a,S}m' \quad &\text{if} \quad m \overset{a,S+\{a\}}{\rightsquigarrow} m' \text{ or } m \overset{\tau,S}{\rightsquigarrow} m'. \end{aligned}$$

These relations are finite in number because we know that only finitely many names are used and only finitely many different S appear on the edges of our graphs. □

We have now provided a notion of regularity for asynchronous processes which allows much more expressivity than the standard notion of regularity for CCS. We have also shown that a suitable notion of bisimulation equivalence is polynomial time decidable over this class of processes. Unfortunately though, this enhanced notion of regularity is not as robust as we would like. In particular, it is the case that one can form parallel compositions and restrictions of CCS regular terms and stay within the class of regular processes [9, 10]. Sadly, this is not the case in the present work. Whilst parallel composition preserves finiteness of the models of regular terms, the restriction of such graphs does not. In fact, using the familiar argument of reducing bisimulation equivalence to the halting problem for two-counter Minsky Machines [11] we can show that allowing restriction of regular terms, unsurprisingly, entails undecidability of our equivalence.

We conclude this section by briefly mentioning that the direct interpretation of asynchronously regular CCS terms as resource graphs can be extended to whole of *aCCS* in such a way as to ensure that Proposition 5.1, (iv) still holds. This extension is non-trivial however and involves defining both the recursion and restriction operators on graphs as the least fixed point of certain functionals so that the resulting resource graphs may become infinite.

6 Conclusion

We have presented a novel approach to modelling asynchronous systems. The chief feature of these new models is the treatment of asynchronous transmission as the use of resources. Resource graphs yield a direct presentation of asynchronous behaviour, without recourse to various commutativity axioms. They also provide a compact representation of many infinite state systems, thereby allowing effective procedures for deciding bisimilarity. We discovered that the somewhat unorthodox notion of asynchronous bisimilarity arises naturally in the category of resource graphs and provided insightful characterisations of this equivalence.

The present work is concerned with synchronising processes rather than communicating processes, that is, no information is transmitted by output actions. Therefore a treatment of asynchrony in the π-calculus is beyond the scope of resource graphs as presented. An issue worth further investigation is a generalisation of the resource graph model which could cater for name passing and dynamic scoping as can be found in the π-calculus.

References

[1] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. In U. Montanari and V.Sassone, editors, *Proceedings CONCUR 96,* Pisa, volume 1119 of *Lecture Notes in Computer Science*, pages 147–162. Springer-Verlag, 1996.

[2] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA, Sophia-Antipolis, 1991.

[3] S. Christensen, Y. Hirshfield, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1993.

[4] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. ACM-POPL*, 1996.

[5] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimulation equivalence of normed basic parallel processes. In *Proc. Mathematical Structures in Computer Science*, 1996.

[6] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proc. ECOOP 91, Geneve*, 1991.

[7] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation and open maps. In *Proceedings* 8^{th} *Annual Symposium on Logic in Computer Science*, pages 418–427. IEEE Computer Society Press, 1993.

[8] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[9] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[10] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[11] F. Moller. Infinite results. In U. Montanari and V.Sassone, editors, *Proceedings CONCUR 96,* Pisa, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer-Verlag, 1996.

[12] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[13] B. Pierce and D. Turner. Pict: A programming language based on the π-calculus, 1996. Univeristy of Cambridge.

[14] P. Selinger. First-order axioms for asynchrony. In M. Bednarczyk, editor, *Proceedings CONCUR 97,* Warsaw, volume 1243 of *Lecture Notes in Computer Science*, pages 376–390. Springer-Verlag, 1997.

[15] C. Stirling. Bisimulation, model checking and other games, 1997. Notes for Mathfit Instructional Meeting on Games and Computation, University of Edinburgh.

[16] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 4*, pages 1–148. Oxford University Press, 1995.