Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 209 (2008) 173–186

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Expressiveness of Process Algebras

Joachim Parrow [1]

Department of Information Technology
Uppsala University
Uppsala, Sweden

Abstract

We examine ways to measure expressiveness of process algebras, and recapitulate and compare some related results from the literature.

Keywords: Process algebra, expressiveness

1 Introduction

The field of process algebras is sometimes looked upon as a jungle of interrelated but separate theories. There are process algebras for expressing nondeterminism, parallelism, distribution, localities, real time, stochastic phenomena, etc, and each of these aspects can be described in different ways. Various researchers study various subcalculi, and our knowledge of the relationship between them is sporadic.

People outside the field sometimes recoil at this diversity, perceiving it as an indication of immaturity and a failure to identify the essentials. A comparison with the lambda calculus and computability theory is inevitable, and then it seems process algebra falls short in defining a single theory of parallel computation. Instead we have lots of them, maybe so many that our customers become confused and do not know what specimen to bring back. In spite of all our time in this business how come we have not brought more order and know which of our formalisms is best, or at least exactly how they interrelate?

The question is, can and should our jungle be turned into a nicely organized garden where there are no weeds and each item has a place that is fully understood? If that is our goal, what are we doing about it?

[1] Email: joachim.parrow@it.uu.se

doi:10.1016/j.entcs.2008.04.011

2 Absolute Expressiveness

In order to appraise a process algebra it is natural to ask what can be expressed in it. We shall use the term "absolute expressiveness" for something that answers this question without referring back to other algebras. Since the semantics almost always is intended to capture the dynamic behaviour of terms the immediate question is "what behaviours are expressible." In other words, we ask what is the set of denotations of closed terms. The problem then lies in finding a denotational model that precisely captures the notion of "behaviour".

But the set of definable behaviours is not all there is to expressiveness. A fundamental idea in computer science is that systems are built hierarchically; construction proceeds by combining simpler systems into larger ones. Therefore we need a set of operations to represent ways a large system can be constructed from its components. The operators of the process algebra are intended to represent basic such operations, and these operations can themselves be combined into more complex ones. To measure expressiveness the question that needs answering is "what operations on behaviours are expressible." In other words, we ask what is the set of denotations of open terms or contexts in the algebra.

For the wide variety of modern process algebras involving mobility, time, probability, locality etc. there seem to be no results on absolute expressiveness. But for the more traditional algebras there are a few results, perhaps because the algebras involved are simpler and the denotations, if not given a complete mathematical formulation, are better understood. Briefly put, they are just transition systems where the transitions carry a label signifying the synchronization event that takes place when the transition is taken. We shall in the following refer to those algebras as *basic* process algebras. So, exactly what transition systems, and what operators on them, are expressible in a given basic algebra?

2.1 Expressiveness of Terms

The terms in a basic process algebra are rooted labeled directed graphs where nodes represent states and labeled edges represent transitions. One might suppose that a minimal requirement of a reasonable algebra is that all such graphs can be denoted, but the real situation is a bit more complicated. First, we are probably only interested in computable graphs, for some well defined notion of computability (a simple one is that the transition relation viewed as a set is computable). Second, process algebras come with a variety of equivalences which in some ways are thought of as identity of behaviour, so it is reasonable to only require that graphs are expressible up to that equivalence. And then it will matter exactly what the equivalence is.

The first to study this kind of problem in technical detail was de Simone (1985) [5]. He worked in the synchronous process algebras MEIJE and SCCS. These share two important characteristics that set them apart from many other algebras. First, the parallel composition operator is of a synchronous kind, in the sense that components composed in parallel execute in lock-step. In other words, an action from a composite system always involves actions from all components. Second, communications may involve any number of components (rather than just one sender and one receiver). Both these characteristics borrow intuition from hardware construction, where components are distinct physical parts.

de Simone's result is that in MEIJE and SCCS all recursively enumerable transition graphs are expressible up to a variant of bisimulation equivalence. Here it should be noted that the semantics of MEIJE and SCCS is not computable; due to the presence of unguarded recursion it is even undecidable if a term has a transition, and de Simone's result makes critical use of this. It is somewhat unsatisfactory to allow non computable primitives in an algebra that intends to convey the essence of executable behaviour, especially if used to derive an expressiveness result.

Another result along these lines is by Baeten, Bergstra and Klop (1987) [1]. They use the basic process algebra ACP, which in many respects is similar to CCS, equipped with around 25 axioms stating which terms are deemed equivalent. Instead of an operational semantics a graph model is explicitly constructed, and through a clever and general diagonalization argument it is shown that there are non-denotable computable graphs. The main result however is in the extension ACP$_\tau$, which adds the unobservable τ-action and axioms for it. The axioms are similar to those for weak bisimulation equivalence. It is shown that ACP$_\tau$ can denote any computable graph up to weak bisimulation. The idea is to define a system of processes that precisely simulates the Turing machine computing the transition relation; all the internal computation steps in this simulation are rendered insignificant in view of the fact that weak bisimulation takes little account of internal transitions.

Vaandrager (1992) [17] gives a nice generalization and exposition of the negative result. He shows that if there are no unobservable actions, then no process algebra with an effective semantics (i.e., the set of transitions from each term is computable) can denote all computable graphs up to trace equivalence. Vaandrager also demonstrates a simple process algebra that can denote all computable graphs up to weak bisimulation equivalence, thus simplifying the proof mentioned in the paragraph above.

In a simple algebra having only the static operators of parallel and interlinking as primitives I show (1989) [12,13] that with only three simple constants all finite state behaviours are definable up to weak bisimulation equivalence. The three constants only have one or two states each, representing the ideas of synchronization, alternation and choice. The result says that it is possible to construct any finite state behaviour if given an unbounded supply of these constants by combining them in parallel. Probably, adding a fourth constant representing unbounded memory (like a stack) would make it possible to duplicate the construction of Baeten et al mentioned above, to express any computable behaviour.

2.2 Expressible Operators

Again, the first results on expressible operators are due to de Simone (1985) [5]. He characterized the expressible operators using structural operational semantics. This is the normal format for giving semantics to the operators of a process algebra and is represented as a system of inference rules for transitions. De Simone showed that, up to bisimulation equivalence, the contexts of MEIJE can express exactly those n-ary operators *op* that have an operational semantics with a finite set of inference rules of the following kind:

$$\frac{\forall i \in S : P_i \xrightarrow{\alpha_i} P'_i \quad R(\alpha_1, \ldots, \alpha_k, \beta)}{op(P_1, \ldots, P_n) \xrightarrow{\beta} T}$$

Here S is a subset of $\{1, \ldots, n\}$ containing k elements, R a $k+1$-ary recursively enumerable relation on action labels, and T any term that contains each P'_i for $i \in S$ and each P_i for $i \in \{1, \ldots, n\} \setminus S$ at most once. Since the P_i, α_i and P'_i are implicitly universally quantified meta variables the rule is completely determined by S, R and T. It says that if the subset S of the operands of *op* can perform actions that satisfy R, then $op(\cdots)$ can perform an action resulting in T: a term built from the derivatives of the operands that acted in the premise, and the rest of the operands that did not act. This format of inference rules is now known as the "de Simone format", and practically all operators in basic process algebras adhere to it.

The main idea of de Simone's construction is that for each operator a kind of supervisor term is designed. The operator instantiated with terms for its operands then behaves exactly as the supervisor in parallel with the same operands. For this to work the parallelism must be synchronous, so that the supervisor so to speak can regulate each action from each operand.

A few years later (1989) I analyzed the expressible operators of the algebra mentioned at the end of Section 2.1 [12]. Here the parallel composition is asynchronous, giving the usual interleaving semantics, and this means that the constructions of de Simone cannot be used. Nevertheless the result is that a significant subset of the de Simone format can be expressed up to weak bisimulation, namely those de Simone operators that satisfy the so called idling rules

$$\frac{\forall i \in S : P_i \longrightarrow P'_i}{op(P_1, \ldots, P_n) \longrightarrow op(P_1^{(\prime)}, \ldots, P_n^{(\prime)})}$$

Here $\longrightarrow$ represents an internal transition without any external communication events, and $P_i^{(\prime)}$ is P'_i for $i \in S$ and P_i for $i \notin S$. The rules says that if a subset of the operands idle, i.e. do nothing externally observable, then the whole term can also idle. This is an effect of asynchronous parallelism: in that setting it is not possible for the supervisor term to prevent an operand from idling. Several usual process algebra operators do not satisfy this, for example the prefix operator and choice operator of ordinary CCS. So in this way it can be argued that prefix and choice cannot be implemented by parallelism alone. Interestingly, if weak trace equivalence is used instead of weak bisimulation both prefix and choice can be defined, but there are still operators that cannot be defined, for example synchronous parallel.

An excellent related exposition is in the paper by Vaandrager [17]. He rephrases the results by de Simone using a simpler synchronous process algebra and gives a more precise definition of what it means to express an operator.

All these results on expressiveness of operators assume that each communication event may involve an arbitrary number of components. This feature is needed in the construction of the supervisory term and its interactions with operands. But process algebras usually have only binary communication. So, to conclude this section, it is remarkable that for most algebras, including the original CCS, there is yet no absolute measure on the expressible operators.

3 Relative Expressiveness

If we cannot determine the expressiveness in an absolute sense we may still obtain relative results. Suppose we have two process algebras **A** and **B**. Say that **A** is *as expressive as* **B** to mean that **A** can express anything that **B** can. Without actually going into details on what is being expressed this can be demonstrated by an encoding $\mathcal{E} : \mathbf{B} \rightarrow \mathbf{A}$ (here and in the following we use **A** and **B** also to represent the carriers of the algebras). The intuition is that whatever an element $b \in \mathbf{B}$ expresses is also expressed by $\mathcal{E}(b)$.

Of course not any mapping $\mathcal{E}$ will suffice for this intuition to work. After all, probably both **A** and **B** are enumerable, so some some mapping will always exist between them. We are here interested in encodings that satisfy two kinds of criteria: They should "preserve reasonably much of the semantics" and they should be defined "in terms of the structure" of **B**. The former guarantees that $\mathcal{E}(b)$ really encodes the intended meaning of b, and the latter ensures that the contexts of **B**, which represent the expressible operators on behaviours, also can be represented in **A**. Several ideas have been suggested to interpret these requirements formally, and often combinations of the ideas are used. We shall here examine some of the most popular.

3.1 Equivalence

If both **A** and **B** are equipped with a behavioural equivalence $\simeq \subseteq (\mathbf{A} \cup \mathbf{B}) \times (\mathbf{A} \cup \mathbf{B})$, defined in exactly the same way on both algebras to convey identity of behaviour, a simple definition is that the encoded term is equivalent to its encoding:

$$\forall b \in \mathbf{B}. \quad \mathcal{E}(b) \simeq b$$

This definition is quite natural and if it holds then $\mathcal{E}$ certainly must be said to "preserve semantics", even though there might still be a choice of which equivalence to use. The problem here is rather that the requirement is too severe. Suppose that we wish to compare the expressiveness of algebras where processes have different sets of observable events (for example involving monadic versus polyadic data transfers) and that the involved equivalence refers to these events. Then the encoding will not be equivalent, simply because it gives rise to observables of different kinds. But it could still be said to express the same thing, in an intuitive sense, even if these are formally expressed with different objects. For example, a polyadic data transfer could be expressed by a sequence of monadic transfers. We therefore need to look at at less strict requirements.

3.2 Preserving Observables

A natural idea is to look for the manifestations of behaviour which are common to processes in both algebras, and require that the encoding preserves them. So choose a set of "observables" $\mathbf{O}$ to represent such manifestations. There are several examples of different kinds of such, for example

- The channels along which a process may communicate
- The possibilities to diverge or converge
- The traces (sequences of communication events, with or without internal actions
- The barbs (Communication capabilities after a sequence of reductions)
- The tests that the process may (or must) pass, for som formal notion of test

Once such a set has been identified to capture relevant aspects of behaviour we can define a function $\mathcal{O} : \mathbf{A} \cup \mathbf{B} \longrightarrow \mathbf{O}$, assigning observables to each element of $\mathbf{A}$ and $\mathbf{B}$, and the criterion for $\mathcal{E}$ to preserve the semantics then is

$$\forall b \in \mathbf{B}. \quad \mathcal{O}(\mathcal{E}(b)) = \mathcal{O}(b)$$

When this is used there is naturally the question if the chosen observables are really exhaustive. Of course, for different purposes different observables are relevant. Also, some of these observables are clearly not enough to capture all of the intended semantics, and therefore this criterion is often combined with other, notably operational correspondence.

3.3 Operational Correspondence

The intuition behind an operational correspondence between b and $\mathcal{E}(b)$ is that both algebras $\mathbf{A}$ and $\mathbf{B}$ are equipped with a reduction relation $\longrightarrow \subseteq (\mathbf{A} \times \mathbf{A}) \cup (\mathbf{B} \times \mathbf{B})$, and that $\mathcal{E}$ preserves the essentials of it. Here $c \longrightarrow c'$ means that c can evolve to c' without interaction with its environment. In many algebras this is represented by the action τ and the reduction is written $\xrightarrow{\tau}$.

In one direction the preservation of semantics is easy: Any reduction in $\mathbf{B}$ should be present in $\mathbf{A}$, though one reduction step may correspond to several steps in the encoding:

$$\forall b, b' \in \mathbf{B}. \quad b \longrightarrow b' \text{ implies } \mathcal{E}(b) \longrightarrow^* \simeq \mathcal{E}(b')$$

Here $\longrightarrow^*$ is the reflexive transitive closure of $\longrightarrow$. The criterion says that that any reduction by b must be mimicked by $\mathcal{E}(b)$, up to a behavioural equivalence $\simeq$.

The converse direction is less straightforward and here one finds a spectrum of possibilities, geared towards particular situations. Clearly just inverting the implication in the criterion above would be insufficient since that would not exclude unwanted reductions in $\mathbf{A}$ to processes that are not encodings of any process in $\mathbf{B}$. A general criterion, though often hard to prove, is that any sequence of reductions from an encoding should be the initial part of a corresponding reduction in $\mathbf{B}$:

$$\forall b \in \mathbf{B}, a \in \mathbf{A}. \quad \mathcal{E}(b) \longrightarrow^* a \text{ implies } \exists b' \in \mathbf{B}. \quad b \longrightarrow^* b' \text{ and } a \longrightarrow^* \simeq \mathcal{E}(b')$$

An operational correspondence can hardly on its own be said to establish that $\mathcal{E}$ preserves the semantics since there may be significants aspects not covered by reductions. Sometimes an operational correspondence is extended to include labelled actions; the definition then becomes highly dependent on the particular process algebras.

3.4 Full Abstraction

This idea requires an equivalence $\simeq \subseteq (\mathbf{A} \times \mathbf{A}) \cup (\mathbf{B} \times \mathbf{B})$ which represents the fact that processes are behaviourally equivalent. The definition of $\simeq$ on $\mathbf{A}$ can be different from the definition on $\mathbf{B}$; this makes full abstraction a very general and widely applicable technique.

The full abstraction criterion is that equivalent processes are mapped to equivalent processes:

$$\forall b, b' \in \mathbf{B}. \quad b \simeq b' \text{ iff } \mathcal{E}(b) \simeq \mathcal{E}(b')$$

This guarantees that the encoding treats behaviourally equivalent processes similarly. Note that the "iff" is implication in both directions. The corresponding forward and backward implications are sometimes referred to as "soundness" and "completeness".

If there is no obvious such equivalence at hand one can often be constructed from a set of observables $\mathbf{O}$, containing possibly different observables from $\mathbf{A}$ and $\mathbf{B}$. Define $\simeq_A$ as the largest congruence in $\mathbf{A}$ that respects observables, namely

$$\forall a, a' \in \mathbf{A}. \quad a \simeq_A a' \quad \text{if} \quad \forall C_A \in \text{contexts of } \mathbf{A}. \quad \mathcal{O}(C_A(a)) = \mathcal{O}(C_A(a'))$$

and similarly

$$\forall b, b' \in \mathbf{B}. \quad b \simeq_B b' \quad \text{if} \quad \forall C_B \in \text{contexts of } \mathbf{B}. \quad \mathcal{O}(C_B(b)) = \mathcal{O}(C_B(b'))$$

Here a context C is just an open term with exactly one free variable, and $C(c)$ is the term obtained by substituting c for the variable. Now we can define $\simeq$ as $\simeq_A \cup \simeq_B$ and apply the definition of full abstraction, it becomes

$$\begin{array}{l} \forall C_B \in \text{contexts of } \mathbf{B}. \quad \mathcal{O}(C_B(b)) = \mathcal{O}(C_B(b')) \\ \text{iff} \\ \forall C_A \in \text{contexts of } \mathbf{A}. \quad \mathcal{O}(C_A(\mathcal{E}(b))) = \mathcal{O}(C_A(\mathcal{E}(b'))) \end{array}$$

This criterion says that processes with the same observables in any context will after translation still have the same observables in any context.

3.5 Weak Full Abstraction

In one sense the criterion of full abstraction can be regarded as too strong. Suppose that $\mathbf{A}$ is some low level formalism encoding a higher level formalism $\mathbf{B}$, by providing for each primitive of $\mathbf{B}$ a protocol in $\mathbf{A}$— think of $\mathcal{E}$ as a compiler from a high level formalism to a low level one. Now, when $\mathcal{E}$ translates from **B** to **A** the resulting terms will always be combinations of such protocols, or if you like, snippets of compiled code. But **A** might contain other things which are not translations of terms in **B**. Although these will never surface in translations they do surface in the contexts of **A**, thereby affecting the definition of full abstraction. It may be unreasonable to ask that the encodings $\mathcal{E}(b)$ and $\mathcal{E}(b')$ behave the same in all possible contexts of **A**, since they will never be exposed to the full range of such contexts. They will only be exposed to those contexts which are encodings of contexts in **B**.

With this insight the definitions above can be relaxed as follows:

$$\forall a, a' \in \mathbf{A}. \quad a \simeq_A a' \quad \text{if} \quad \forall C_B \in \text{contexts of } \mathbf{B}. \quad \mathcal{O}(\mathcal{E}(C_B)(a)) = \mathcal{O}(\mathcal{E}(C_B)(a'))$$

Here terms in **A** are equivalent if they behave the same in all encoded contexts (so the definition depends on $\mathcal{E}$). If $\mathcal{E}(C_B)(\mathcal{E}(b)) = \mathcal{E}(C_B(b))$ then weak full abstraction becomes

$$\forall C_B \in \text{contexts of } \mathbf{B}. \quad \mathcal{O}(C_B(b)) = \mathcal{O}(C_B(b'))$$

$$\text{iff}$$

$$\forall C_B \in \text{contexts of } \mathbf{B}. \quad \mathcal{O}(\mathcal{E}(C_B(b))) = \mathcal{O}(\mathcal{E}(C_B(b')))$$

In some cases the encodable contexts can be defined through a type system in **A**, and then weak full abstraction becomes the same as full abstraction of the well-typed fragment of **A**.

Variants of full abstraction use more limited forms of contexts. For example "for all contexts of **B**" can be replaced by "for all tests in **B**", for some well defined notion of test. There is then still a difference between "all tests in **A**" and "all tests that are encodings of tests in **B**", giving rise to a difference between full abstraction and weak full abstraction.

In conclusion there are many ways to interpret the criterion "$\mathcal{E}$ preserves reasonable parts of the semantics," and these are sometimes related. For example, the condition of equivalence will imply full abstraction for the same equivalence and might imply preservation of some observables. Many of these relations are dependent on the particular algebras under study, for example in how different choices of equivalence affect the results. In all there is, unfortunately, little discernible general structure to the choices of criteria.

3.6 Structural Definition

If there is no requirement on $\mathcal{E}$ that it is structurally defined over the operators in **B**, then the encoding is only relevant for expressiveness of terms. For expressiveness of operators additional requirements are needed, and they come in a few variants.

3.6.1 Homomorphy

The strongest criterion is that $\mathcal{E}$ is homomorphic for some chosen operators. Formally $\mathcal{E}$ is homomorphic on the operator *op* if *op* is present in both **A** and **B**, and

$$\forall b \in \mathbf{B}. \quad \mathcal{E}(op(b)) = op(\mathcal{E}(b))$$

The definition generalizes to operators of higher arity in the obvious way. Naturally $\mathcal{E}$ cannot be homomorphic on all operators (or it would become identity). But if **A** and **B** share an operator and this operator is intended to represent the same construction then homomorphy is reasonable. A prime example is to require homomorphy for parallel composition; this entails that $\mathcal{E}$ respects the distribution of a term into parallel components exactly.

3.6.2 Compositionality

A weaker criterion is that an operator is translated compositionally: this means that for each operator in **B** there is a corresponding context in **A** that implements $\mathcal{E}$. If this holds for all operators we say that $\mathcal{E}$ is compositional:

$$\forall op \in \mathbf{B}. \quad \exists C \in \text{ contexts in } \mathbf{A}. \quad \forall b \in \mathbf{B}. \quad \mathcal{E}(op(b)) = C(\mathcal{E}(b))$$

again with the obvious extension to operators of higher arity. This implies that any context in **B** can be represented as a context in **A**.

3.6.3 Weak Compositionality

Sometimes compositionality is hard to achieve where a slightly weaker version is possible, allowing an single outermost context. We say that $\mathcal{E}$ is weakly compositional if

$$\exists C \in \text{ contexts in } \mathbf{A}. \quad \forall b \in \mathbf{B}. \quad \mathcal{E}(b) = C(\mathcal{F}(b))$$

where $\mathcal{F} : \mathbf{B} \rightarrow \mathbf{A}$ is compositional. Again, thinking of $\mathcal{E}$ as a compiler clarifies the issue: here C can be regarded as the run time system and $\mathcal{F}$ as the compiler proper. The relationship with compositionality (i.e. under which circumstances a weakly compositional encoding implies the existence of a compositional one) has not been studied in any detail and might be an interesting avenue to explore.

3.7 Examples: The Asynchronous π-calculus

In the last ten years there has been a significant effort to relate various process algebras, in particular for the π-calculus, its subcalculi and extensions. We shall here briefly look at a few of these results. This is not intended to be an exhaustive list but will give the reader some idea of typical problems and solutions.

The asynchronous π-calculus ($a\pi$) is a subcalculus of the π-calculus where there are no continuations following the output primitive. This means that it is not possible to say "first output this and then do that." Unguarded outputs can be regarded as messages in transit, where their eventual reception will not directly affect the sender. Also, in most versions of $a\pi$ there is no choice operator.

When $a\pi$ was introduced (independently by Honda and Tokoro 1991 and by Boudol 1992) it was claimed that it is as expressive as the full π-calculus, at least without the choice operator. The motivation is a compositional encoding— actually it is homomorphic on all operators except prefix — which introduces a small handshake protocol that signals the sender when a message has arrived. Variants of this protocol has been analyzed in several papers.

Quaglia and Walker (2000) [15] use a variant which encodes the polyadic choice-free π-calculus into monadic $a\pi$, and prove that the encoding is weakly fully abstract with respect to barbed congruence, one of the standard equivalences in the π-calculus. They further define a type system to filter out the π-calculus terms that are not encodings of $a\pi$ terms, and obtain full abstraction for the well typed calculus.

Cacciagrano, Corradini and Palamidessi (2006) [4] derive a seemingly contradictory result: no encoding of even the monadic π-calculus into monadic $a\pi$ which is compositional for the prefix operator can be weakly fully abstract for must-tests. Note that a requirement for this negative result is that the encoding is compositional for prefixes. Thus this does not say anything about inexpressible terms, rather it says something about inexpressible contexts. The reason it is not actually contradicting the result by Quaglia and Walker is that must tests are more discriminating than barbs, in particular when it comes to distinguishing between terms with different divergence potentials.

Earlier, Nestmann and Pierce studied encodings of the choice operator into monadic $a\pi$ (2000) [8]. The idea is to introduce a lock as an explicit critical resource and translate choice into parallel where a parallel branch must claim the lock to proceed, therefore only one branch will actually progress. The encoding only works for input-guarded choice, i.e., a choice where the first action in each branch is an input guard. It is compositional, and homomorphic in all operators except choice. The encoding is proved weakly coupled equivalent (a version of weak bisimulation), and preserves divergence properties. A variant of the encoding does not preserve divergence but is equivalent for ordinary weak bisimulation. So there is a trade-off in that different encodings preserve different parts of the semantics. Other related encodings of various forms of choice are demonstrated by Nestmann (2000) [7].

Palamidessi (1997) [9,10] has also studied the encoding of choice and has an interesting negative result: an encoding of the full π-calculus, with mixed choice (i.e. where branches of a choice can be guarded by both input and output actions) into $a\pi$, or even into the the π-calculus restricted to separate choice (i.e. where all choices are either between input guards or between output guards) is impossible for an encoding that is compositional, homomorphic in parallel and preserving the observables of maximal traces. The latter condition actually implies that divergence potentials are preserved. The proof is by a reduction to the distributed leader election protocol. Choosing a branch is roughly the same as choosing a leader. A fully distributed solution to leader election always introduce a divergence possibility, so any encoding must also introduce a possibility to diverge. Comparing Palamidessi's and Nestmann's results there seems to be a sharp line between separate choice and mixed choice: one can be encoded while preserving parallelism and convergence properties, and the other cannot. There are still unsettled questions, e.g. if mixed choice can be encoded preserving observation equivalence (which does not take account of divergence).

3.8 Examples: Limited Prefixes

The prefix operator is the source of dynamic behaviour since it introduces communication actions. I have showed (2000) [14] that for the choice-free π-calculus with replication, the subcalculus where prefix is nested to depth at most three is equally expressive. The encoding works by replacing every operator in a term by a trio: three nested prefixes where the first and last are used for communication with other trios and the middle performs the work of the operator. It is weakly compositional and equivalent for weak bisimulation. So this result really is about expressiveness for terms rather than contexts: it says that no fewer terms are definable when prefix is limited to depth three. In the same work I show that limiting prefixes to depth two makes it impossible to find an encoding that is equivalent for weak bisimulation; there is no requirement on compositionality for the negative result so it is about undefinable terms (and not contexts). There is a yet unproved conjecture that the result extends to the π-calculus with choice if the trios also include choice.

The fusion calculus is a generalization of π where actions may trigger side effects. As a measure of its expressive power, Laneve and Victor (1999) [6] have shown that restricting prefix to depth one, i.e. a prefix cannot be followed by anything at all, it is still possible to encode the full fusion calculus. The encoding is compositional and homomorphic for parallel, enjoys a kind of operational correspondence and weak full abstraction with respect to barbs. It works by using the side effects of single actions to enforce temporal sequencing; it is a quite striking result that the very primitive side effects of the fusion calculus can enforce arbitrary such temporal relationships.

Yoshida considers a version of the π-calculus without any prefix operators at all (1998) [18,19]. In their place there is a small number of combinator terms; these do contain prefixes, so in other words prefixes are limited to occur in the combinator terms. It turns out that different sets of combinators can encode different subcalculi of π. For example, with five simple combinators the asynchronous choice free π-calculus can be encoded. This encoding is homomorphic for all operators except input, where it is not even compositional, so the result is about expressible terms rather than contexts. The encoded terms are weakly bisimulation equivalent. The negative results are for encodings which are homomorphic, preserve barbs and have an operational correspondence.

Palamidessi et al (2006) [11] study the subset of the π-calculus where all prefixes are replicated. In other words, all unguarded prefixes are persistent in the sense that they will never cease to offer interaction possibilities. It is shown that there is no encoding of π into this subcalculus which is homomorphic for parallel and weakly fully abstract for barbed congruence. The proof idea is that the equation $X|X \simeq X$ holds in the subcalculus.

3.9 Examples: Adding Data Structures

The basic process algebras contain no primitives for data transfer, and the π-calculus allows for transfer only of atomic names. But the motivation behind these calculi has always been to model transfer of more complex data structures. Already in the original CCS there was an encoding of data structures, using an infinite choice operator. And the very first article on the π-calculus presented encodings of data structures, without examining their properties formally. The idea is that instead of sending a data object we send a link leading to a copy of the object.

Adding more advanced data objects involves either moving to a higher order formalism (where the data objects transferred are themselves processes in the language) or extending the algebra with additional constructs specifically for data terms. The former approach is taken by Sangiorgi (1993) [16]. He shows that the higher-order π-calculus can be encoded by the standard π-calculus; the encoding is compositional, in fact homomorphic for most operators, fully abstract for barbed congruence and enjoys an operational correspondence.

The latter approach, of defining data structures separately, was taken by Baldamus, Victor and myself (2004) [2,3]. We studied the problem of encoding data enriched calculi into the ordinary π-calculus. Interestingly there is here a trade off between different kinds of encodings. One is compositional and weakly fully abstract for may tests, the other is weakly compositional and fully abstract for may tests. It is still not known if there exists a compositional and fully abstract encoding. The main difficulty is that a process may receive a data term and subsequently deconstruct it and test the parts for equality, this is why the proofs for the higher order encoding do not carry over to this case. Briefly put, the problem is the π-calculus processes that are not encodings. They can either be excluded (giving weak full abstraction) or protected against by an outermost context (giving weak compositionality).

4 Concluding Remarks

This brief survey of expressiveness has shown that great care has to be taken when formulating and interpreting results: expressiveness can be taken to mean many different things, and just saying "**A** is as expressive as **B**" without further qualifying what is meant is not very informative.

We have hardly answered the questions from the introduction: Should we transform the jungle into a garden, and are we getting close? We seem farther from it than ever. There are n different process algebras, where n is finite but growing unboundedly. We attempt to remedy the situation by comparing the different algebras in a systematic way. But we end up with k different ways of comparing their expressiveness. This gives us a potential for $n \times n \times k$ distinct expressiveness results. Is this really the road to eventual success?

In actual fact the jungle and garden analogy is not very accurate and the comparison with the lambda calculus is misleading. The jungle, if there is one, is in all the information processing systems being constructed, using the inventiveness of millions of programmers and system builders. Our process algebras form neither jungle nor garden; they are not part of nature and they are not a decoration to please the eye. They are tools and there to do a job. And the job is immense. The information technology industry is today building the most complicated artifacts ever invented by humanity. So we shall keep developing the tools and gain as much confidence as we can in their relationships and operating parameters, but the goal to produce one all purpose tool just isn't there.

Once upon a time a man went into a hardware store. Approaching the clerk he said "How do you do, I need to cut some small piece of wood and have been told that I could probably make good use of a knife. I understand you can sell me one, is this correct?" The clerk replied "Certainly sir, we have a wide selection here" and pointed to a cabinet full of knives of different sizes, manufacturers and colours, exhibiting a variety of sharpness, durability and craftsmanship. To her surprise the man recoiled and cried out "What do you mean by all this? I just want the one knife, and of course I want the best knife. This abundance just shows me you have not yet found out which knife is best, despite all your years in the business."

References

[1] Jos C. M. Baeten, Jan A. Bergstra, and Jan Willem Klop. On the consistency of koomen's fair abstraction rule. *Theor. Comput. Sci.*, 51:129–176, 1987.

[2] Michael Baldamus, Joachim Parrow, and Björn Victor. Spi calculus translated to pi–calculus preserving may-tests. In *LICS*, pages 22–31. IEEE Computer Society, 2004.

[3] Michael Baldamus, Joachim Parrow, and Björn Victor. A fully abstract encoding of the *pi*-calculus with data terms. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 1202–1213. Springer, 2005.

[4] Diletta Cacciagrano, Flavio Corradini, and Catuscia Palamidessi. Separation of synchronous and asynchronous communication via testing. *Electr. Notes Theor. Comput. Sci.*, 154(3):95–108, 2006.

[5] Robert de Simone. Higher-level synchronising devices in meije-sccs. *Theor. Comput. Sci.*, 37:245–267, 1985.

[6] Cosimo Laneve and Björn Victor. Solos in concert. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 513–523. Springer, 1999.

[7] Uwe Nestmann. What is a good encoding of guarded choice? *Inf. Comput.*, 156(1-2):287–319, 2000.

[8] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. *Inf. Comput.*, 163(1):1–69, 2000.

[9] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In *POPL*, pages 256–265, 1997.

[10] Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[11] Catuscia Palamidessi, Vijay A. Saraswat, Frank D. Valencia, and Björn Victor. On the expressiveness of linearity vs persistence in the asychronous pi-calculus. In *LICS*, pages 59–68. IEEE Computer Society, 2006.

[12] Joachim Parrow. The expressive power of simple parallelism. In Eddy Odijk, Martin Rem, and Jean-Claude Syre, editors, *PARLE (2)*, volume 366 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 1989.

[13] Joachim Parrow. The expressive power of parallelism. *Future Generation Computer Systems*, 6:271–285, 1990.

[14] Joachim Parrow. Trios in concert. In *Proof, Language and Interaction, Essays in Honour of Robin Milner*, pages 621–637. MIT Press, 2000.

[15] Paola Quaglia and David Walker. On synchronous and asynchronous mobile processes. In Jerzy Tiuryn, editor, *FoSSaCS*, volume 1784 of *Lecture Notes in Computer Science*, pages 283–296. Springer, 2000.

[16] Davide Sangiorgi. From pi-calculus to higher-order pi-calculus - and back. In Marie-Claude Gaudel and Jean-Pierre Jouannaud, editors, *TAPSOFT*, volume 668 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 1993.

[17] Frits W. Vaandrager. Expressive results for process algebras. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 666 of *Lecture Notes in Computer Science*, pages 609–638. Springer, 1992.

[18] Nobuko Yoshida. Minimality and separation results on asynchronous mobile processes: Representability theorems by concurrent combinators (extended abstract). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 1998.

[19] Nobuko Yoshida. Minimality and separation results on asynchronous mobile processes - representability theorems by concurrent combinators. *Theor. Comput. Sci.*, 274(1-2):231–276, 2002.