Math. Struct. in Comp. Science (2016), *vol.* 26, *pp.* 639–654. © Cambridge University Press 2014
doi:10.1017/S0960129514000279 First published online 13 November 2014

Full abstraction for expressiveness: history, myths and facts[†]

DANIELE GORLA[‡] and UWE NESTMANN[§]

[‡]*Dip. di Informatica, 'Sapienza' Università di Roma. Via Salaria 113, 00198 Roma, Italy*
Email: gorla@di.uniroma1.it
[§]*Technische Universitat Berlin. Ernst-Reuter-Platz 7, 10587 Berlin, Germany*
Email: uwe.nestmann@tu-berlin.de

Received 7 October 2012; revised 7 February 2014

What does it mean that an encoding is fully abstract? What does it *not* mean? In this position paper, we want to help the reader to evaluate the real benefits of using such a notion when studying the expressiveness of programming languages. Several examples and counterexamples are given. In some cases, we work at a very abstract level; in other cases, we give concrete samples taken from the field of process calculi, where the theory of expressiveness has been mostly developed in the last years.

1. Introduction

The notion of *full abstraction* came into existence as an attempt to answer the question 'What is the meaning of a program (part)?', which also drove the development of the field of denotational semantics. A denotation is a mathematical object that should capture the 'abstract' meaning of programs. A denotation function is a mapping from one world (the one of programs) into another world (the one of denotations). The 'concrete' meaning of programs is typically formalized via an operational semantics of the language. However, as the operational semantics is usually too concrete in that it distinguishes programs that may abstractly mean the same (e.g. they produce the same outputs for the same inputs), it is common practice to associate with it an observational equivalence that relates two programs whenever they operationally behave the same in every execution context. The aim of full abstraction was to reconcile the two views (Plotkin 1977): it requires that two programs (or program parts) have the same denotations precisely when they are observationally equivalent. The denotational semantics is then called *fully abstract* for the respective observational equivalence. In summary, the notion of full abstraction, originally, was conceived to achieve a quite specific goal in programming language semantics: precisely characterizing denotational models. It was not meant to be used as a tool to analyse the relative expressive power of programming languages.

The need to *compare programming formalisms* w.r.t. their expressive power dates back to the very origins of computer science, e.g., with the formal comparison of Turing machines, λ-calculi and partially recursive functions. Such a need has become more dramatic with the

[†] Supported by the Deutsche Forschungsgemeinschaft, grant NE-1505/2-1.

proliferation of programming languages and primitives for synchronization. Indeed, the community soon realized that Turing completeness was not enough, since almost every formalism could encode Turing machines and proposed alternative ways to compare different formalisms; some pioneering works are Chandra and Manna (1976), Landin (1966), Lipton (1975), Lipton *et al.* (1974), Paterson and Hewitt (1970), Reynolds (1970), Reynolds (1981) and Steele and Sussman (1976). In essence, however, most comparisons were carried out by just mutually encoding the formalisms into each other.

Maybe simply triggered by the idea of using mappings between formalisms, some years later a number of researchers started to use the notion of full abstraction to study the expressive power of programming languages (Mitchell 1993). Indeed, *encodings* are another way of moving from one world (made up by terms of the source language) into another (made up by terms of the target language) (Riecke 1991; Shapiro 1991; de Boer and Palamidessi 1994). See also Ph.D. thesis of Perez (2009), which contains a comprehensive overview on many of these and the above-mentioned works. Then, by mimicking the approach of classical semantics, the notion of full abstraction was put forward as a – if not *the* – criterion for the correctness of encodings. Probably, this was also justified by the fact that programming languages, unlike arbitrary computational formalisms, are syntactic formalisms that are usually equipped with notions of equivalence. With this view, a correct encoding has to map equivalent source terms into equivalent target terms and, conversely, equivalent images of this mapping must have originated from equivalent source terms. One of the benefits of this approach relies in the possibility of transferring equations back and forth; for example, this allows one to use tools of the target language to prove equivalences in the source language.

In concurrency theory, examples of contributions that refer to full abstraction as a decisive criterion to measure the relative expressive power of languages are by Amadio (2000), Boreale (1998), Fournet and Gonthier (1996), Merro (1998), Sangiorgi (1993),Victor and Parrow (1996) and others. For example, Fournet and Gonthier (1996) state '*we assess the relative expressive powers of miscellaneous calculi from the existence of fully-abstract encodings between them*'. Depending on the individual case at hand, though, the respective authors more or less implicitly suggest that full abstraction shall be accompanied by some property like compositionality as a requirement on the encoding, or by other results like operational correspondence.

In comparison to the original denotational approach, the role of denotations in the context of encodings between programming languages is played by equivalence classes in the target. While in the original setting the domain of denotations was meant to yield more abstract representations of the meaning of programs, this is no more true in the encoding setting: here, the target of the encoding is a language itself, so an additional equivalence relation is generally required to abstract away from the superfluous details.

Differently from the denotational approach, where the target model is to capture source-equivalent programs within just one mathematical object, the encoding approach resembles compilers mapping higher-level programs into lower-level (assembly) code. Not surprisingly, such an encoding typically imposes a protocol among the translated components. As a matter of fact, it often occurs that the target model allows for observers (i.e., contexts) of the translated programs that do not respect the expected protocol.

Consequently, these observers can easily interfere with and thereby break equivalences between encoded terms (see Section 5 for an extensive example). As a result, it is usually very difficult to develop encodings that enjoy full abstraction w.r.t. reference equivalences without constraining their contextual properties; instead, full abstraction is then only stated w.r.t. translated contexts, which respect the protocol by definition.

Not being explicit about further requirements on encodings, like syntactic or structural criteria, it may at first be surprising to see the number of immediate – but useless – full abstraction results that one then gets trivially for free (cf. Section 4). It is also possible to construct less trivial, though still useless, results by involving actual encoding constructions. The problem that can be identified behind these examples is that the choice of admitted equivalences in the source and target language is not properly constrained. In fact, it seems that it can hardly be: no proper principle has yet been identified to characterize equivalences that are admitted or intended in full abstraction results. Finally, it may be surprising that full abstraction results between two languages cannot be formally compared, for example, by stating that one result is stronger than another one (cf. Section 5.3).

In the light of the above-mentioned problems, we come to the conclusion (like others, notably Beauxis *et al.* (2008)) that full abstraction as *the* criterion to study and measure the relative expressive power of programming languages is largely debatable. In this paper, we provide a variety of examples that may convince the reader that – from the expressiveness point of view – full abstraction alone is not informative enough concerning the actual quality of an encoding. It is nice to have, when it can be proved for some equivalences, but it is *not* the ultimate measure.

The problem of expressiveness has been identified as an increasingly important topic in concurrency theory (Nestmann 2006; Parrow 2008). In recent years, new approaches for the evaluation of the quality of encodings from the expressiveness perspective have been proposed (Palamidessi 2003; Carbone and Maffeis 2003; Gorla 2008; Haagensen *et al.* 2008; Gorla 2010a,b; Peters and Nestmann 2012; van Glabbeek 2012; Fu and Lu 2010). This paper can also be seen as an *a posteriori* justification for such works.

2. Technical preliminaries

2.1. *Basic notions*

We consider the relative expressiveness of *models*, for which we regard triples composed of: a language $\mathcal{L}$, often generated via a BNF-grammar inducing an algebraic signature; the operational semantics $\longmapsto \subseteq \mathcal{L} \times \mathcal{L}$, specified via *reductions* (e.g., β-reductions in λ-calculus, synchronizations in CCS, communications in π-calculus, ...); and an equivalence relation $\simeq \subseteq \mathcal{L} \times \mathcal{L}$. As full abstraction focuses on the equivalences, from now on we shall mostly ignore the reduction relation and consider models just as pairs of the form $\mathbf{M} = (\mathcal{L}, \simeq)$.

An *encoding* $[\![\cdot]\!]$ of model $\mathbf{S} = (\mathcal{L}_\mathbf{S}, \simeq_\mathbf{S})$ into model $\mathbf{T} = (\mathcal{L}_\mathbf{T}, \simeq_\mathbf{T})$ is a (total) function $[\![\cdot]\!] : \mathcal{L}_\mathbf{S} \longrightarrow \mathcal{L}_\mathbf{T}$ mapping elements of $\mathcal{L}_\mathbf{S}$ into elements of $\mathcal{L}_\mathbf{T}$; by overloading, we also

write $[\![\cdot]\!] : \mathbf{S} \longrightarrow \mathbf{T}$. We sometimes abbreviate $\mathcal{L}_\mathbf{S}$ and $\mathcal{L}_\mathbf{T}$ by $\mathcal{S}$ and $\mathcal{T}$. We let S and T range over terms of the source language ($\mathcal{S}$) and target language ($\mathcal{T}$), respectively.

Definition 1. *An encoding* $[\![\cdot]\!] : \mathbf{S} \to \mathbf{T}$ *is* fully abstract *iff, for every* $S_1, S_2 \in \mathcal{L}_\mathbf{S}$*:*

$$(S_1 \simeq_\mathbf{S} S_2) \iff ([\![S_1]\!] \simeq_\mathbf{T} [\![S_2]\!]).$$

The encoding $[\![\cdot]\!] : \mathcal{L}_\mathbf{S} \longrightarrow \mathcal{L}_\mathbf{T}$ *is then called* fully abstract w.r.t. $(\simeq_\mathbf{S}, \simeq_\mathbf{T})$.

Usually, the '$\Rightarrow$' implication is called *equivalence preservation* or *completeness* (as all equivalent terms are captured via the endoing by the target equivalence), whereas the '$\Leftarrow$' implication is called *equivalence reflection* or *soundness* (as all of the terms captured by the target equivalence are actually intended to be equivalent in the source).

If the language $\mathcal{S}$ is generated by some grammar, then the induced algebraic structure *may* be used to define the encoding inductively. Likewise, one may then formulate specific criteria like compositionality. Note that, in general, the induced algebraic signatures of $\mathcal{S}$ and $\mathcal{T}$ (if both are generated in a structured way) are different. Thus, we cannot easily refer to the standard algebraic terminology of homomorphisms in order to establish criteria on encodings. This is, however, sometimes done for individual operators that are present in both $\mathcal{S}$ and $\mathcal{T}$; for process calculus models, the homomorphism requirement applied to the parallel operator is a prominent example and constitutes an important, although not universally agreed, building block for many separation results (Palamidessi 2003; Peters and Nestmann 2012; Peters *et al.* 2013). Note that full abstraction itself does not at all refer to structural requirements on encodings.

2.2. *Process calculi in a nutshell*

As several of our examples involve process calculi, we provide some very basic and intuitive notions on them. More precisely, we focus on variants of the π-calculus; for full details, we refer the interested reader to the standard textbooks on the subject, e.g., Milner (1999) and Sangiorgi and Walker (2001). The syntax of π-calculus is specified by the following BNF:

$$P ::= \mathbf{0} \mid P \mid P \mid (\nu\, a)P \mid !P \mid \tau.P \mid a(x).P \mid \overline{a}\langle b\rangle.P \mid P + P. \tag{1}$$

Intuitively, the intended semantics is as follows:

— $\mathbf{0}$ is the dead process.
— $P_1 \mid P_2$ is the parallel composition of processes P_1 and P_2; the components P_1 and P_2 may behave independently, but they may also synchronize via a binary handshake, called *communication* (see below).
— $(\nu\, a)P$ defines name a to be local for P, i.e., name a is different from any other name occurring outside P.
— $!P$ is the replication operator, providing as many parallel copies of P as desired.
— $\tau.P$, $a(x).P$ and $\overline{a}\langle b\rangle.P$ are action prefixed processes, with $\overline{a}\langle b\rangle$ being the complementary action of $a(x)$. Process $\tau.P$ evolves in isolation, without engaging in a communication, leading to P. By contrast, $a(x).P$ and $\overline{a}\langle b\rangle.P$ cannot evolve alone, but only when put in parallel: in such a case, $a(x).P_1 \mid \overline{a}\langle b\rangle.P_2$ generates a reduction that consumes

both prefixes, leading to $P_1\{^b/_x\} \mid P_2$. Thus, π-calculus's communication is binary and nominal, i.e., fully defined by the top-level occurrence of unstructured names.

— $P_1 + P_2$ is the non-deterministic choice between P_1 and P_2; the choice is mutually exclusive, in the sense that, if P_1 is selected for execution, then P_2 is lost for ever.

In this paper, we also sometimes use the *polyadic* and the *asynchronous* versions of the π-calculus (see Milner (1993); Boudol (1992); Honda and Tokoro (1991)). The former has the same syntax as in (1), with the only difference that in the polyadic version actions have multiple arguments, i.e., $a(x_1, \ldots, x_n)$ and $\overline{a}\langle b_1, \ldots, b_n\rangle$, for some $n \geqslant 0$. The latter has the following syntax:

$$P ::= \mathbf{0} \mid P \mid P \mid (\nu\, a)P \mid !P \mid \tau.P \mid a(x).P \mid \overline{a}\langle b\rangle. \tag{2}$$

There are two essential aspects of the asynchronous π-calculus: (1) The absence of continuations after outputs: a process that sends a message has no built-in possibility to have a continuation behaviour directly depend on the successful reception of this message; such a behaviour would have to be realized with an explicit communication protocol, using acknowledgement messages. (2) The absence of choice: as the asynchronous interpretation of the output particle $\overline{a}\langle b\rangle$ represents the instantaneous emission of datum b along channel a, processes like $\overline{a}\langle b\rangle + P$ would counter the interpretation of (guarded) choice. Thus, in the asynchronous π-calculus, choices are usually omitted (like in our case) or limited to only input-guarded processes.

2.3. *Equivalences and congruence properties*

A common practice in the formal study of programming languages is the specification of when two programs are equivalent, in the sense that they *behave in the same way*. Among all the possible behavioural equivalences, the most interesting ones exhibit *congruence properties*, i.e., equivalences that are preserved by (execution) contexts that can be formulated within the language.

More formally, a *context* $\mathcal{C}[\_]$ is a term with a single occurrence of a *hole*, denoted as '_', which acts as a placeholder for an arbitrary other term. If we replace the hole in context $\mathcal{C}[\_]$ with term P, we obtain the term $\mathcal{C}[P]$. An equivalence $\simeq$ is a congruence w.r.t. a set of contexts, if $P \simeq Q$ implies $\mathcal{C}[P] \simeq \mathcal{C}[Q]$ for every considered context $\mathcal{C}[\_]$.

In concurrency theory, as well as in programming languages, one of the main concerns is to define 'canonical' notions of congruence on top of the so-called reduction bisimulation (which we do not recall here, as it is not needed for the understanding of the paper). This is usually done by fixing an obvious observation predicate (sometimes called 'barb') and requiring that two terms must exhibit the same observables along all reductions and in *any* context. Such a canonical construction yields the notion of *barbed congruence* (Milner and Sangiorgi 1992), of which we assume the weak version (weak reductions, weak barbs) as the reference equivalence $\cong$ for the process models in this paper.

Deviating from barbed congruence, which requires that equivalent terms still remain equivalent in *any* context, it sometimes suffices to use a limited form of congruence obtained by reducing the set of considered contexts. For example, in process calculi, one is also interested in congruences w.r.t. just parallel contexts, i.e., in equivalences that are just

preserved by contexts of the form '$\_ \mid P$'; such contexts suffice to model the execution environment of a process. Because of the reduced observational power, the resulting congruence is of course weaker than the general full congruence.

3. Informative full abstraction results

Representative for the literature on full abstraction results, we explain just some examples in more detail in order to exhibit the variety of possible observations. What we are going to present are some of the *true positives* of our study, i.e., encodings where full abstraction holds ('positives') and where it should be so because they convey informative expressiveness results, which are expected or 'intuitively agreed-upon' ('true').

One of the pioneering full abstraction results is Mitchell (1993). There, the notion of full abstraction w.r.t. an equivalence (very similar in spirit to barbed congruence) is introduced as the reference criterion for the correctness of encodings and is applied to study some variants of functional languages. For example, it is proved that a `let` construct can be encoded in a fully abstract way in the untyped λ-calculus, or that recursive types can be encoded in non-recursive ones in the setting of the λ-calculus. Again, concerning fully abstract encodings of different λ-calculi, (Riecke 1991) proves that the call-by-name and the lazy version can be encoded in the call-by-value one, that in turn can be encoded in the lazy one. By using different criteria (mostly adapted from the work on the comparison of formal systems done by Kleene (1952)), Felleisen (1991) proves that call-by-name and call-by-value are incomparable, i.e., none can be expressed in the other.

In the field of process calculi, an interesting full abstraction result is found in Nestmann and Pierce (2000). There, input-guarded choices in the asynchronous π-calculus are rendered in the choice-free calculus with an encoding that allows to directly equate – w.r.t. the *asynchronous* barbed congruence (Amadio *et al.* 1998), usually considered the reference equivalence for such calculi – terms and their translations. As a by-product, full abstraction comes for free in this case. However, this encoding introduces divergence: it may turn a terminating term into a non-terminating one. This can be seen as a further argument to support the claim that full abstraction cannot be the only criterion for assessing the quality of an encoding. Indeed, it is widely accepted in the field of distributed computing that divergence may well matter when trying to simulate one construct in another. A notable example is Herlihy (1991), where it is proved that the atomic effect of the `test-and-set` primitive cannot be simulated by any sequence of `read` and `write` in a concurrent system with shared variables without introducing divergence. In the field of concurrency theory, divergence is sometimes neglected: e.g., weak barbed congruence ignores it (for example, $!\tau \cong \mathbf{0}$ in π-calculus). For this reason, an encoding that is fully abstract w.r.t. weak barbed congruence is often well accepted, even if it introduces divergence. For example, Fournet and Gonthier (1996) define a process calculus, called *Join*, that can encode π-calculus and can be encoded in it in a fully abstract way w.r.t. their reference equivalences; in doing this, divergence is totally ignored and the encoding is in fact not divergence-free. By contrast, Nestmann and Pierce (2000) do not take this position: they also provide a divergence-free encoding between the asynchronous π-calculus with input-guarded choices and the choice-free calculus; however, the encoding

is fully abstract only w.r.t. *coupled simulation*, an equivalence that is strictly coarser than weak barbed congruence.

Further, full abstraction results in process calculi are found in Merro and Sangiorgi (2004), in which the expressive power of a dialect of the asynchronous π-calculus, called Lπ, is studied. Mainly, Lπ differs from the asynchronous π-calculus by imposing that received names cannot be used by the receiver for performing inputs. There are several interesting results. (1) Lπ allows to encode polyadic communications into monadic ones in a fully abstract way (Merro 2000), by a straightforward adaptation of the (nonfully abstract) encoding by Milner presented in Section 5. (2) The benefits of full abstraction are exploited for establishing a characterization of barbed congruence, which would otherwise be much more difficult to prove, via a labelled bisimilarity. In particular, Lπ is encoded into its subcalculus with only restricted outputs (i.e., output particles that emit restricted names), called Lπ_I, by adapting the encoding from Boreale (1998). Then, it is proved that the encoding is fully abstract w.r.t. barbed congruence for Lπ and a bisimulation-based equivalence for Lπ_I. Thus, to prove equivalences in the first language, it is enough to work in the target, where an easier-to-handle equivalence is developed. A similar approach is taken in the *higher-order π-calculus*, a variant of the π-calculus where also processes (and not only names) can be passed in a communication. Sangiorgi (1993) proved that higher-order communications can be encoded into first-order ones in a fully abstract way; this fact can be used to simplify the way in which equivalences are proved in the higher-order setting.

4. Useless full abstraction results

We now show that there are 'bad' encodings that nevertheless satisfy full abstraction; these play the role of the *false positives* of our study and are used to show that such a property is not that demanding as it is sometimes believed.

We start our discussion with an example taken from Beauxis *et al.* (2008), where the expressive power of two well-known models, viz. Turing machines and finite automata, are compared.

Fact 1. *Let* TM *and* FA *denote the sets of Turing machines and of finite automata. Let* $\mathbf{TM} = (\mathrm{TM}, \simeq_{\mathrm{TM}})$ *and* $\mathbf{FA} = (\mathrm{FA}, \simeq_{\mathrm{FA}})$ *be the models defined with their standard language equivalences. Then, there exists a fully abstract encoding* $[\![\cdot]\!] : \mathbf{TM} \to \mathbf{FA}$.

Proof. Consider an enumeration of Turing machines, $\{T_n\}_n$, and an enumeration of minimal finite automata, $\{A_n\}_n$. Consider the following encoding of Turing machines into (minimal) finite automata:

$$\begin{array}{lll} [\![T_m]\!] & = & [\![T_k]\!] \qquad \text{if } k < m \text{ and } T_k \simeq_{\mathrm{TM}} T_m \\ [\![T_m]\!] & = & A_n \qquad \text{otherwise} \end{array}$$

where n is the minimum number such that A_n has not been used to encode any T_k, with $k < m$. By definition, we have that, for every m and n, $T_m \simeq_{\mathrm{TM}} T_n$ if and only if $[\![T_m]\!] \simeq_{\mathrm{FA}} [\![T_n]\!]$. □

Of course, even if the encoding provided in Fact 1 enjoys full abstraction, this certainly does not prove that finite automata are as powerful as Turing machines. Obviously, the encoding is non-effective. This is fine for our purpose, which is simply to show that full abstraction alone, i.e., without extra conditions on the encoding, is not a very meaningful notion. It would be even more interesting to exhibit an effective and fully abstract encoding between some **S** and **T** for which most people would agree that **T** is strictly less powerful than **S**.

The following observation highlights that also the trivial encoding – mapping every source term to the same target term – can be considered fully abstract.

Fact 2. *Let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary. Let $[\![\cdot]\!] : \mathcal{S} \to \mathcal{T}$ with $[\![S]\!] = T$, for all $S \in \mathcal{S}$ and some $T \in \mathcal{T}$. Let $\mathbf{S} = (\mathcal{S}, \mathcal{S} \times \mathcal{S})$ and $\mathbf{T} = (\mathcal{T}, \simeq_{\mathbf{T}})$, for arbitrary $\simeq_{\mathbf{T}}$. Then, $[\![\cdot]\!] : \mathbf{S} \to \mathbf{T}$ is fully abstract.*

Proof. (Case '$\Rightarrow$'): for every $S_1 \simeq_{\mathbf{S}} S_2$, it holds that $[\![S]\!]_1 \simeq_{\mathbf{T}} [\![S_2]\!]$: indeed, $[\![S_1]\!] = T = [\![S_2]\!]$ and the implication holds because of reflexivity of $\simeq_{\mathbf{T}}$. (Case '$\Leftarrow$'): for every S_1 and S_2 such that $[\![S_1]\!] \simeq_{\mathbf{T}} [\![S_2]\!]$, it holds that $S_1 \simeq_{\mathbf{S}} S_2$: indeed, $\simeq_{\mathbf{S}}$ is the total relation on $\mathcal{S}$ and hence it relates every pair of source terms. □

It is easy to rule out the above trivially collapsing encoding by some syntactic or structural condition on encodings. However, even then, any encoding can enjoy a full abstraction property. To this aim, let $\mathrm{Ker}(f)$, for some function f, be the relation pairing elements with the same image under f.

Fact 3. *Let $[\![\cdot]\!] : \mathcal{S} \to \mathcal{T}$ be arbitrary. Let $\mathbf{S} = (\mathcal{S}, \mathrm{Ker}([\![\cdot]\!]))$ and $\mathbf{T} = (\mathcal{T}, \mathrm{Id})$. Then, $[\![\cdot]\!] : \mathbf{S} \to \mathbf{T}$ is fully abstract.*

Proof. By definition, $(S_1, S_2) \in \mathrm{Ker}([\![\cdot]\!])$ iff $[\![S_1]\!] = [\![S_2]\!]$, i.e., $([\![S_1]\!], [\![S_2]\!]) \in \mathrm{Id}$. □

The previous results are all an instance of a more general result by Parrow (2014), stating that a fully abstract encoding $[\![\cdot]\!] : \mathbf{S} \to \mathbf{T}$ exists if and only if the cardinality of the $\simeq_{\mathbf{T}}$-quotient is greater than or equal to the cardinality of the $\simeq_{\mathbf{S}}$-quotient, in the sense that there exists an injective mapping from the latter to the former.

5. On changing equivalences and weak full abstraction

Since full abstraction deeply relies on (source/target) equivalences, it is natural to analyse the effect of changing them. In this section, we first show that moving from a sub-model to its super-model does not generally let the naive embedding be fully abstract, because of the richer power arising from the richer set of contexts in the target language. Then, we show that more complex encodings (and also more interesting than naive embeddings) may suffer from the same problems. In this case, it can be worthwhile to consider weaker forms of full abstraction to accept such encodings. This can be achieved by changing the target equivalence to require weaker congruence properties via smaller sets of considered contexts, as we have already hinted at in earlier sections. Finally, we show that changing equivalences in a largely unconstrained way may easily lead to meaningless results.

5.1. *On the power of contexts: naive embeddings*

We consider an example in the setting of the π-calculus. It is very easy to render the asynchronous π-calculus into the synchronous one by placing a '**.0**' after every output particle. Similarly, it is widely accepted that the natural equivalences for such models are the asynchronous weak barbed congruence[†], $\cong_{\mathrm{a}}$, and the standard weak barbed congruence, $\cong$, respectively. Nevertheless, we have the following:

Fact 4. *Consider the encoding of the asynchronous π-calculus with $\cong_{\mathrm{a}}$ into the synchronous π-calculus with $\cong$ such that*

$$[\![\, \overline{a}\langle b\rangle \,]\!] = \overline{a}\langle b\rangle.\mathbf{0}$$

and that acts homomorphically on all the other operators. This embedding does not preserve the reference equivalences.

Proof. Consider $P = \mathbf{0}$ and $Q = a(x).\overline{a}\langle x\rangle$; it is well known (Amadio *et al.* 1998) that $P \cong_{\mathrm{a}} Q$, but $[\![P]\!] \ncong [\![Q]\!]$. □

Remark. The classical non-congruence property of weak bisimilarity in CCS (Milner 1989) also pops up in a naive-embedding setting. Consider the original CCS (Milner 1989) and its sub-language with guarded choice CCS_{gc} (Milner 1999). It is well known (Milner 1989) that weak bisimilarity $\approx$ is *not* a full congruence in CCS; in particular, it is not closed under general non-deterministic choice. Instead, $\approx^{\circ}$, the largest congruence contained in $\approx$, is our reference equivalence in CCS. By contrast, $\approx$ is a full congruence in CCS_{gc} and it is also our reference equivalence for this subcalculus.

Let us consider $\mathbf{M}_1 = (\mathrm{CCS}_{gc}, \approx)$ and $\mathbf{M}_2 = (\mathrm{CCS}, \approx^{\circ})$. Then, the identity language embedding of $\mathbf{M}_1$ into $\mathbf{M}_2$ does not preserve equivalences, as $a.\mathbf{0} \approx a.\tau.\mathbf{0}$, but $a.\mathbf{0} \not\approx^{\circ} a.\tau.\mathbf{0}$ due to the distinguishing general-choice context $(\_ + b.\mathbf{0})$.

The above examples highlight the fact that it is indeed naive to expect of an embedding that keeps source terms (almost) identical will automatically result in a positive full abstraction result. The chosen equivalences are crucial and even the choice of the reference equivalence for a calculus can be harmful for naive encodings. So, the above results are 'false negatives', when naively just focusing on language translation and they are 'true negatives' when taking the contextual differences of the two calculi – via different availability of contexts and induced different granularity of the equivalences – into account.

5.2. *On the power of contexts: weak full abstraction*

We have just discussed the impact of the target contexts to establish full abstraction results. In this section, we use a well-known encoding example to emphasize this phenomenon and point out a wide-spread problem of encodings, which are only correct w.r.t. notions

[†] As longly discussed in Amadio *et al.* (1998), the two notions differ in two facts: (1) in the asynchronous version, one is only allowed to observe outputs; and (2) in the asynchronous version, only contexts of the asynchronous π-calculus are considered.

of congruence that severely limit the set of observing target contexts. As we will see, the limitation on the set of contexts is defined by means of the encoding itself.

Consider the encoding by Milner (1993) that shows how to render the transmission of two names along a channel by multiple single-name exchanges:

$$[\![\, \overline{a}\langle b, c\rangle.P \,]\!] \;=\; (\nu\, d)\overline{a}\langle d\rangle.\overline{d}\langle b\rangle.\overline{d}\langle c\rangle.[\![\, P \,]\!]$$
$$[\![\, a(x, y).Q \,]\!] \;=\; a(z).z(x).z(y).[\![\, Q \,]\!].$$

The idea is that $\overline{a}\langle d\rangle$ sends the new (secret) channel d along channel a; then, $a(z)$ retrieves such a message from a and uses it to replace z in the continuation. Then, the simultaneous transmission of the two names (b and c) is rendered via two single-name transmissions along the new channel d (we have used just two names for simplicity; the case for $n > 2$ names is straightforward).

This looks like a rather natural encoding; however, it does not enjoy full abstraction w.r.t. the most widely-used equivalences for process calculi. For example, we have that $\overline{a}\langle b, c\rangle.\overline{a}\langle b, c\rangle$ is equivalent to $\overline{a}\langle b, c\rangle \mid \overline{a}\langle b, c\rangle$, w.r.t. almost every equivalence (actually, in all the ones that ignore causality). However, their encodings, placed in the context '$- \mid a(z)$' behave differently. Indeed, $[\![\, \overline{a}\langle b, c\rangle.\overline{a}\langle b, c\rangle \,]\!] \mid a(z)$ reduces to $(\nu\, d)\overline{d}\langle b\rangle.\overline{d}\langle c\rangle.[\![\, \overline{a}\langle b, c\rangle \,]\!]$ and the latter term is usually equivalent to a dead process. On the contrary, $[\![\, \overline{a}\langle b, c\rangle \mid \overline{a}\langle b, c\rangle \,]\!] \mid a(z)$ reduces to $(\nu\, d)\overline{d}\langle b\rangle.\overline{d}\langle c\rangle \mid [\![\, \overline{a}\langle b, c\rangle \,]\!]$ and the latter term is usually equivalent to $[\![\, \overline{a}\langle b, c\rangle \,]\!]$, that is not dead. Here the problem is that the context '$- \mid a(z)$' does not respect the protocol put forward by the encoding; in particular, it does not retrieve the two names coming along the received channel d. This fact blocks $[\![\, \overline{a}\langle b, c\rangle.\overline{a}\langle b, c\rangle \,]\!]$ before its completion.

Thus, the polyadic π-calculus can be encoded in the monadic one, but this encoding is *not* fully abstract. Does this mean that the monadic π-calculus is less expressive than the polyadic one, or that the encoding just shown is not 'good'? In the concurrency community, it is very well accepted that the two formalisms have the same expressive power and that Milner's encoding is the most natural and 'good' way to pass from one to the other. A similar situation occurs in the setting of the asynchronous π-calculus: it can encode the synchronous one (Boudol 1992; Honda and Tokoro 1991) but such encodings are not fully abstract (Quaglia and Walker 2000) and it can be proved (Cacciagrano *et al.* 2007) that no fully abstract encoding can exist, if the reference equivalence is must testing (De Nicola and Hennessy 1984). So, these look like examples of 'false negatives' for full abstraction.

There are two possible ways to solve this kind of problems: declare the encoding fully abstract either w.r.t. *encoded* contexts (i.e., the target equivalence is closed only under context that arise from the translation of a source context) or w.r.t. some suitable *typed* equivalence (that only considers contexts that respect the protocol put forward by the encoding). Parrow (2008) introduced the term *weak full abstraction* to describe such kinds of relaxation. The two solutions are similar in spirit; the first one has been adopted, e.g., in Boreale (1998) and Palamidessi *et al.* (2006), whereas the second one in Quaglia and Walker (2005) and Yoshida (1996). The second case might be considered more informative than the first, as it does not directly depend on the encoding itself, but might refer to

some independently useful type system. In both cases, however, the target equivalences are quite weak and the use of target tools is hardly applicable. At the end, these encodings can now be considered examples of 'true positives' for *weak* full abstraction.

5.3. *On changing equivalences, abstractly*

We have just seen that a 'false negative' result can be turned to a 'true positive' one by changing the target equivalence. In this section, we consider the effect of varying the involved equivalences, both at the source and at the target level and show that this can easily lead to unsatisfying results.

The common understanding of full abstraction is that it exactly preserves and reflects equivalences of the source and of the target model. So, it states an exact correspondence of the equivalences of the source and the target. For this reason, it should be expected that, by changing only one of the equivalences involved, the full abstraction result will be broken. Actually, this is the case only in some circumstances.

5.4. *Source only*

First of all, for a fully abstract encoding, one cannot change only the *source* equivalence without breaking full abstraction, be it by weakening or strengthening the equivalence.

Fact 5. *Let* $\mathbf{S} = (\mathcal{S}, \simeq_{\mathbf{S}})$, $\mathbf{T} = (\mathcal{T}, \simeq_{\mathbf{T}})$ *and* $[\![\cdot]\!] : \mathbf{S} \to \mathbf{T}$ *fully abstract w.r.t.* $(\simeq_{\mathbf{S}}, \simeq_{\mathbf{T}})$. *Then,* $[\![\cdot]\!]$ *is not fully abstract w.r.t.* $(\simeq'_{\mathbf{S}}, \simeq_{\mathbf{T}})$, *for every* $\simeq'_{\mathbf{S}} \subset \simeq_{\mathbf{S}}$ *and* $\simeq'_{\mathbf{S}} \supset \simeq_{\mathbf{S}}$.

Proof. If $\simeq'_{\mathbf{S}} \subset \simeq_{\mathbf{S}}$, then at least one $\simeq_{\mathbf{S}}$-equivalence class has been split; this breaks equivalence reflection. If $\simeq'_{\mathbf{S}} \supset \simeq_{\mathbf{S}}$, then at least two different $\simeq_{\mathbf{S}}$-equivalence classes have been merged; this breaks equivalence preservation. □

5.5. *Target only*

By contrast, it is possible to change just the target equivalence without breaking full abstraction only if the encoding is not surjective (as it is usually the case). For surjective encodings, a situation similar to Fact 5 holds.

Fact 6. *Let* $\mathbf{S} = (\mathcal{S}, \simeq_{\mathbf{S}})$, $\mathbf{T} = (\mathcal{T}, \simeq_{\mathbf{T}})$ *and* $[\![\cdot]\!] : \mathcal{S} \to \mathcal{T}$ *fully abstract w.r.t.* $(\simeq_{\mathbf{S}}, \simeq_{\mathbf{T}})$ *and not surjective. Then, there exists a* $\simeq'_{\mathbf{T}}$ *different from* $\simeq_{\mathbf{T}}$ *such that* $[\![\cdot]\!]$ *is fully abstract w.r.t.* $(\simeq_{\mathbf{S}}, \simeq'_{\mathbf{T}})$.

Proof. Let $T \notin Im([\![\cdot]\!])$, let C_T the $\simeq_{\mathbf{T}}$-equivalence class of T and C another $\simeq_{\mathbf{T}}$-equivalence class (at least another one exists). Then, $\simeq'_{\mathbf{T}}$ is obtained by moving T from C_T into C. □

5.6. *Both source and target*

If we are ready to change both the source and the target, it is possible to move from one equivalence to a stricter/coarser one (both in the source and in the target) without affecting

full abstraction. Notice that different source terms usually have different encodings ($[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ is injective); so the hypothesis in Fact 7 is satisfied in practice.

Fact 7. *Let* $\mathbf{S} = (\mathcal{S}, \simeq_{\mathbf{S}})$, $\mathbf{T} = (\mathcal{T}, \simeq_{\mathbf{T}})$ *and* $[\![\cdot]\!] : \mathcal{S} \to \mathcal{T}$ *fully abstract w.r.t.* $(\simeq_{\mathbf{S}}, \simeq_{\mathbf{T}})$ *and injective. Then, for every* $\simeq'_{\mathbf{S}} \subset \simeq_{\mathbf{S}}$, *there exists* $\simeq'_{\mathbf{T}} \subset \simeq_{\mathbf{T}}$ *such that* $[\![\cdot]\!]$ *is fully abstract w.r.t.* $(\simeq'_{\mathbf{S}}, \simeq'_{\mathbf{T}})$.

Proof. If $\simeq'_{\mathbf{S}} \subset \simeq_{\mathbf{S}}$, then at least one $\simeq_{\mathbf{S}}$-equivalence class has been split into several $\simeq'_{\mathbf{S}}$-classes. We obtain $\simeq'_{\mathbf{T}}$ by splitting the corresponding classes accordingly. This can always be done except if we have two different (but $\simeq_{\mathbf{S}}$-equivalent) source terms with the same encoding and we try to separate them. But this is ruled out by the injectivity of the encoding function. □

Fact 8. *Let* $\mathbf{S} = (\mathcal{S}, \simeq_{\mathbf{S}})$, $\mathbf{T} = (\mathcal{T}, \simeq_{\mathbf{T}})$ *and* $[\![\cdot]\!] : \mathcal{S} \to \mathcal{T}$ *fully abstract w.r.t.* $(\simeq_{\mathbf{S}}, \simeq_{\mathbf{T}})$. *Then, for every* $\simeq'_{\mathbf{S}} \supset \simeq_{\mathbf{S}}$, *there exists* $\simeq'_{\mathbf{T}} \supset \simeq_{\mathbf{T}}$ *such that* $[\![\cdot]\!]$ *is fully abstract w.r.t.* $(\simeq'_{\mathbf{S}}, \simeq'_{\mathbf{T}})$.

Proof. If $\simeq'_{\mathbf{S}} \supset \simeq_{\mathbf{S}}$, then at least two $\simeq_{\mathbf{S}}$-equivalence classes have been merged together into one $\simeq'_{\mathbf{S}}$-class. We obtain $\simeq'_{\mathbf{T}}$ by merging the corresponding classes accordingly. □

Two similar results hold even if we first change the target equivalence and then try to find a proper source equivalence that still ensures full abstraction. In this case, the injectivity hypothesis needed in Fact 7 is not needed anymore.

To conclude, it is worth remarking that, if we only admit standard equivalences (for process calculi, consider, e.g., those in van Glabbeek (1990, 1993)), the above proofs do no longer necessarily apply. For example, take Fact 8 and an encoding fully abstract w.r.t. bisimilarities (i.e., $\simeq_{\mathbf{S}}$ and $\simeq_{\mathbf{T}}$ are the bisimilarities for **S** and **T**). If we move to trace equivalence (i.e., $\simeq'_{\mathbf{S}}$ is trace equivalence for **S**, that usually is strictly coarser than bisimilarity), it is not necessarily the case that, by mimicking the construction shown in the proof, we obtain a $\simeq'_{\mathbf{T}}$ that is also a well-known equivalence (in particular, the trace equivalence for **T**).

5.7. *Theory?*

In summary, it seems that nothing systematic can yet be said for full abstraction results obtained by weakening or strengthening the involved equivalences. It would be interesting to understand whether this is a general problem, or whether there are particular settings in which some derivation of full abstraction results becomes possible. Likewise, we also have no formal criterion at hand to state that one full abstraction result for an encoding should be considered preferable over another for the same models.

6. Conclusions

In this paper, we have collected evidence to support the claim that full abstraction is not the right – at least: not a sufficient – criterion for assessing the quality of an encoding. This position contrasts with a common trend in the community, where this criterion has been the reference for around two decades.

To support our position, we have shown that obviously 'bad' encodings may enjoy some form of full abstraction, whereas some obviously 'good' ones do not. We have also made the case that additional requirements like, e.g., effectiveness of the encoding, may be desirable for obtaining more informative results. Moreover, full abstraction behaves strangely w.r.t. the reference equivalences: on the one hand, as expected, changing the equivalences can break full abstraction results; on the other hand, it is also possible to properly change them without affecting the property at all. In the light of these observations, the notion of full abstraction seems rather fragile and should be used carefully.

Another important argument in the discussion on full abstraction for expressiveness arises from the confusion of the notions of encodability and implementability[†]. Here, two fields with different interests come to different conclusions. If one is only interested in (1) the possibility of transferring equations (as implied by the usual reference equivalences) between source and target languages, or one is interested in (2) the implementability of one language into the other (preserving distributability or not introducing divergence), then the criteria for assessing the quality of an encoding are clearly judged differently. In case (1), an informative full abstraction result is needed, but then it had better not be a weak full abstraction result, that only holds for translated contexts. In case (2), full abstraction is simply not needed, as other criteria may take their place.

To conclude, this paper aims at highlighting a problem, not at providing a solution. Some first attempts have appeared in the literature, but a final solution is still missing.

Acknowledgements

We sincerely thank the anonymous referees for their constructive comments, which together much helped to straighten and better convey the message of this document. We are particularly grateful for discussions with Joachim Parrow and his concrete suggestions to improve the structure and message of this paper. We further thank Kirstin Peters for stimulating discussions and proof-reading, and Catuscia Palamidessi for the long-term exchange on the subject and for encouraging us to actually write down our position statement in this form. The original idea for this endeavour was born during the BASICS 2009 Workshop 'Computation and Interaction' at Shanghai Jiao Tong University.

References

Amadio, R. M. (2000) On modelling mobility. *Theoretical Computer Science* **240** 147–176.

Amadio, R. M., Castellani, I. and Sangiorgi, D. (1998) On bisimulations for the asynchronous pi-calculus. *Theoretical Computer Science* **195** (2) 291–324.

Beauxis, R., Palamidessi, C. and Valencia, F. D. (2008) On the asynchronous nature of the asynchronous pi-calculus. In: Concurrency, Graphs and Models. *Springer Lecture Notes in Computer Science* **5065** 473–492.

[†] In the case of process calculi, the quest for *distributed* implementability often yields different results.

Boreale, M. (1998) On the expressiveness of internal mobility in name-passing calculi. *Theoreatical Computer Science* **195** (2) 205–226.

Boudol, G. (1992) Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis.

Cacciagrano, D., Corradini, F. and Palamidessi, C. (2007) Separation of synchronous and asynchronous communication via testing. *Theoretical Computer Science* **386** (3) 218–235.

Carbone, M. and Maffeis, S. (2003) On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing* **10** (2) 70–98.

Chandra, A. K. and Manna, Z. (1976) On the power of programming features. *Computer Languages* **1** (3) 219–232.

de Boer, F. S. and Palamidessi, C. (1994) Embedding as a tool for language comparison. *Information and Computation* **108** (1) 128–157.

De Nicola, R. and Hennessy, M. (1984) Testing equivalences for processes. *Theoretical Computer Science* **34** 83–133.

Felleisen, M. (1991) On the expressive power of programming languages. *Science of Computer Programming* **17** (1–3) 35–75.

Fournet, C. and Gonthier, G. (1996) The reflexive chemical abstract machine and the join-calculus. In: *Proceedings of Principles of Programming Languages (POPL)*, ACM 372–385.

Fu, Y. and Lu, H. (2010) On the expressiveness of interaction. *Theoretical Computer Science* **411** (11–13) 1387–1451.

Gorla, D. (2008) Comparing communication primitives via their relative expressive power. *Information and Computation* **206** (8) 931–952.

Gorla, D. (2010a). A taxonomy of process calculi for distribution and mobility. *Distributed Computing* **23** (4) 273–299.

Gorla, D. (2010b) Towards a unified approach to encodability and separation results for process calculi. *Information and Computation* **208** (9) 1031–1053.

Haagensen, B., Maffeis, S. and Phillips, I. (2008) Matching systems for concurrent calculi. In: Proceedings of International Workshop on Expressiveness of Concurrency (EXPRESS). *Electronic Notes in Theoretical Computer Science* **194** (2) 85–99.

Herlihy, M. (1991) Wait-free synchronization. *ACM Transactions on Programming Languages and Systems* **13** (1) 124–149.

Honda, K. and Tokoro, M. (1991) An object calculus for asynchronous communication. In: Proceedings of European Conference on Object-Oriented Programming (ECOOP'91). *Springer Lecture Notes in Computer Science* **512** 133–147.

Kleene, S. C. (1952) *Introduction to Metamathematics*, Van Nostrand, New York.

Landin, P. J. (1966) The next 700 programming languages. *Communications of the ACM* **9** 157–166.

Lipton, R. J. (1975) Reduction: A new method of proving properties of systems of processes. In: *Proceedings of Principles of Programming Languages (POPL)*, ACM 78–86.

Lipton, R. J., Snyder, L. and Zalcstein, Y. (1974) A comparative study of models of parallel computation. In: *15th Annual Symposium on Switching and Automata Theory*, IEEE 145–155.

Merro, M. (1998) On the expressiveness of Chi, update, and fusion calculi. In: Palamidessi, C. and Castellani, I. (eds.) Proceedings of International Workshop on Expressiveness of Concurrency (EXPRESS '98). *Electronic Notes in Theoretical Computer Science* **16**(2), Elsevier Science Publishers 133–144.

Merro, M. (2000) Locality and polyadicity in asynchronous name-passing calculi In: Proceedings of Foundations of Software Science and Computation Structures (FoSSaCS). *Springer Lecture Notes in Computer Science* **1784** 238–251.

Merro, M. and Sangiorgi, D. (2004) On asynchrony in name-passing calculi. *Mathematical Structures in Computer Science* **14** (5) 715–767.

Milner, R. (1989) *Communication and Concurrency*, Prentice Hall.

Milner, R. (1993) The polyadic π-calculus: A tutorial. In: *Logic and Algebra of Specification*, Series F: Computer and System Sciences, volume 94, Springer.

Milner, R. (1999) *Communicating and Mobile Systems: The π-Calculus*, Cambridge University Press.

Milner, R. and Sangiorgi, D. (1992) Barbed bisimulation In: Proceedings of International Colloquium on Automata, Languages and Programming (ICALP). *Springer Lecture Notes in Computer Science* **623** 685–695.

Mitchell, J. (1993) On abstraction and the expressive power of programming languages. *Science of Computer Programming* **21** (2) 141–163.

Nestmann, U. (2006) Welcome to the jungle: A subjective guide to mobile process calculi In: Proceedings of International Conference on Concurrency Theory (CONCUR). *Springer Lecture Notes in Computer Science* **4137** 52–63.

Nestmann, U. and Pierce, B. C. (2000) Decoding choice encodings. *Information and Computation* **163** (1) 1–59.

Palamidessi, C. (2003) Comparing the expressive power of the synchronous and asynchronous Pi-calculi. *Mathematical Structures in Computer Science* **13** (5) 685–719.

Palamidessi, C., Saraswat, V. A., Valencia, F. D. and Victor, B. (2006) On the expressiveness of linearity vs persistence in the asychronous Pi-calculus. In: *Proceedings of Logic in Computer Science (LICS)*, IEEE Computer Society 59–68.

Parrow, J. (2008) Expressiveness of process algebras. *Electronic Notes in Theoretical Computer Science* **209** 173–186.

Parrow, J. (2014) General conditions for full abstraction. In this issue of *Mathematical Structures in Computer Science*.

Paterson, M. S. and Hewitt, C. E. (1970) Comparative schematology. In: *Conference on Concurrent Systems and Parallel Computation*, ACM 119–127.

Perez J. A. (2009) *Higher-Order Concurrency: Expressiveness and Decidability Results*, Ph.D. thesis, University of Bologna.

Peters, K. and Nestmann, U. (2012) Is it a 'good' encoding of mixed choice? In: Birkedal, L. (ed.) Foundations of Software Science and Computation Structures (FoSSaCS). *Springer Lecture Notes in Computer Science* **7213** 210–224.

Peters, K., Nestmann, U. and Goltz, U. (2013) On distributability in process calculi. In: Felleisen, M. and Gardner, P. (eds.) European Symposium on Programming (ESOP). *Springer Lecture Notes in Computer Science* **7792** 310–329.

Plotkin, G. D. (1977) LCF considered as a programming language. *Theoretical Computer Science* **5** (3) 223–255.

Quaglia, P. and Walker, D. (2000) On synchronous and asynchronous mobile processes. In: Proceedings of Foundations of Software Science and Computation Structures (FoSSaCS). *Springer Lecture Notes in Computer Science* **1784** 283–296.

Quaglia, P. and Walker, D. (2005) Types and full abstraction for polyadic *pi*-calculus. *Information and Computation* **200** (2) 215–246.

Reynolds, J. C. (1970) GEDANKEN – a simple typeless language based on the principle of completeness and the reference concept. *Communications of the ACM* **13** 308–319.

Reynolds, J. C. (1981) *The Essence of ALGOL*, North Holland 345–372.

Riecke, J. G. (1991) Fully abstract translations between functional languages. In: *Proceedings of Principles of Programming Languages (POPL)*, ACM 245–254.

Sangiorgi, D. (1993) *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*, Ph.D. thesis, Laboratory for Foundations of Computer Science, University of Edinburgh. CST-99-93 (also published as ECS-LFCS-93-266).

Sangiorgi, D. and Walker, D. (2001) *The π-Calculus: A Theory of Mobile Processes*, Cambridge University Press.

Shapiro, E. Y. (1991) Separating concurrent languages with categories of language embeddings (extended abstract). In: *Proceedings of Symposium on Theory of Computing (STOC)*, ACM 198–208.

Steele, G. L. and Sussman, G. J. (1976) Lambda: The ultimate imperative. AI Lab Memo AIM-353, MIT AI Lab.

van Glabbeek, R. (2012) Musing on encodings and expressiveness. In: Proceedings of EXPRESS/SOS. *Electronic Proceedings in Theoretical Computer Science* **89** 81–98.

van Glabbeek, R. J. (1990) The linear time-branching time spectrum (extended abstract) In: Proceedings of International Conference on Concurrency Theory (CONCUR). *Springer Lecture Notes in Computer Science* **458** 278–297.

van Glabbeek, R. J. (1993) The linear time - branching time spectrum ii. In: Proceedings of International Conference on Concurrency Theory (CONCUR). *Springer Lecture Notes in Computer Science* **715** 66–81.

Victor, B. and Parrow, J. (1996) Constraints as processes In: Montanari, U. and Sassone, V. (eds.) Proceedings of International Conference on Concurrency Theory (CONCUR '96). *Springer-Verlag Lecture Notes in Computer Science* **119** 389–405.

Yoshida, N. (1996) Graph types for monadic mobile processes. In: Proceedings of Foundations of Software Technology and Theoretical Computer Science (FSTTCS). *Springer Lecture Notes in Computer Science* **1180** 371–386.