EPTCS 190

Proceedings of the

Combined 22th International Workshop on Expressiveness in Concurrency

and 12th Workshop on Structural Operational Semantics

Madrid, Spain, 31st August 2015

Edited by: Silvia Crafa and Daniel E. Gebler

Published: 26th August 2015
DOI: 10.4204/EPTCS.190
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Table of Contents i

Preface ii

Comparing Deadlock-Free Session Typed Processes 1
Ornela Dardha and Jorge A. Pérez

On Compensation Primitives as Adaptable Processes 16
Jovana Dedeić, Jovanka Pantović and Jorge A. Pérez

SOS rule formats for convex and abstract probabilistic bisimulations 31
Pedro R. D'Argenio, Matias David Lee and Daniel Gebler

Analysing and Comparing Encodability Criteria 46
Kirstin Peters and Rob van Glabbeek

Encoding CSP into CCS 61
Meike Hatzel, Christoph Wagner, Kirstin Peters and Uwe Nestmann

Encoding the Factorisation Calculus 76
Reuben N. S. Rowe

Preface

This volume contains the proceedings of the Combined 22nd International Workshop on Expressiveness in Concurrency and the 12th Workshop on Structural Operational Semantics (EXPRESS/SOS 2015) which was held on 31 August 2015 in Madrid, Spain, as an affiliated workshop of CONCUR 2015, the 26th International Conference on Concurrency Theory.

The EXPRESS workshops aim at bringing together researchers interested in the expressiveness of various formal systems and semantic notions, particularly in the field of concurrency. Their focus has traditionally been on the comparison between programming concepts (such as concurrent, functional, imperative, logic and object-oriented programming) and between mathematical models of computation (such as process algebras, Petri nets, event structures, modal logics, and rewrite systems) on the basis of their relative expressive power. The EXPRESS workshop series has run successfully since 1994 and over the years this focus has become broadly construed.

The SOS workshops aim at being a forum for researchers, students and practitioners interested in new developments, and directions for future investigation, in the field of structural operational semantics. One of the specific goals of the SOS workshop series is to establish synergies between the concurrency and programming language communities working on the theory and practice of SOS. Reports on applications of SOS to other fields are also most welcome, including: modelling and analysis of biological systems, security of computer systems programming, modelling and analysis of embedded systems, specification of middle-ware and coordination languages, programming language semantics and implementation, static analysis software and hardware verification, semantics for domain-specific languages and model-based engineering.

Since 2012, the EXPRESS and SOS communities have organized an annual combined EXPRESS/SOS workshop on the expressiveness of mathematical models of computation and the formal semantics of systems and programming concepts.

We received ten full paper submissions out of which the programme committee selected six for publication and presentation at the workshop. These proceedings contain these selected contributions. The workshop had an invited presentation:

Bisimulation techniques in probabilistic higher-order languages,
by Davide Sangiorgi (Universita' di Bologna)

We would like to thank the authors of the submitted papers, the invited speaker, the members of the programme committee, and their subreviewers for their contribution to both the meeting and this volume. We also thank the CONCUR 2015 organizing committee for hosting EXPRESS/SOS 2015. Finally, we would like to thank our EPTCS editor Rob van Glabbeek for publishing these proceedings and his help during the preparation.

Silvia Crafa and Daniel E. Gebler,
August 2015

Program Committee

Johannes Borgstrom (Uppsala University, Sweden)
Matteo Cimini (Indiana University, Bloomington, Indiana)
Silvia Crafa (University of Padova, Italy)
Pedro R. D'Argenio (University of Cordoba, Argentina)
Daniel Gebler (VU University Amsterdam, The Nederlands)

Thomas Given-Wilson (Inria, France)
Thomas T. Hildebrandt (IT University of Copenhagen, Denmark)
Daniel Hirschkoff (ENS Lyon, France)
Stefan Milius (University of Erlangen-Nurnberg, Germany)
Mohammad R. Mousavi (Halmstad University, Sweden)
Kirstin Peters (Technical University of Berlin, Germany)
Damien Pous (ENS Lyon, France)
Irek Ulidowski (University of Leister, United Kingdom)

Additional Reviewers

Massimo Bartoletti, Paul Brunet, Cinzia Di Giusto, Carla Ferreira, Daniele Gorla, Barry Jay, Johannes Aman Pohjola.

Comparing Deadlock-Free Session Typed Processes

Ornela Dardha
University of Glasgow, United Kingdom

Jorge A. Pérez
University of Groningen, The Netherlands

Besides respecting prescribed protocols, communication-centric systems should never "get stuck". This requirement has been expressed by liveness properties such as progress or (dead)lock freedom. Several typing disciplines that ensure these properties for mobile processes have been proposed. Unfortunately, very little is known about the precise relationship between these disciplines–and the classes of typed processes they induce.

In this paper, we compare $\mathscr{L}$ and $\mathscr{K}$, two classes of deadlock-free, session typed concurrent processes. The class $\mathscr{L}$ stands out for its canonicity: it results naturally from interpretations of linear logic propositions as session types. The class $\mathscr{K}$, obtained by encoding session types into Kobayashi's usage types, includes processes not typable in other type systems.

We show that $\mathscr{L}$ is strictly included in $\mathscr{K}$. We also identify the precise condition under which $\mathscr{L}$ and $\mathscr{K}$ coincide. One key observation is that the *degree of sharing* between parallel processes determines a new expressiveness hierarchy for typed processes. We also provide a type-preserving rewriting procedure of processes in $\mathscr{K}$ into processes in $\mathscr{L}$. This procedure suggests that, while effective, the degree of sharing is a rather subtle criteria for distinguishing typed processes.

1 Introduction

The goal of this work is to formally relate different type systems for the π-calculus. Our interest is in *session-based concurrency*, a type-based approach to communication correctness: dialogues between participants are structured into *sessions*, basic communication units; descriptions of interaction sequences are then abstracted as *session types* [12] which are checked against process specifications. We offer the first formal comparison between different type systems that enforce *(dead)lock freedom*, the liveness property that ensures session communications never "get stuck". Our approach relates the classes of typed processes that such systems induce. To this end, we identify a property on the structure of typed parallel processes, the *degree of sharing*, which is key in distinguishing two salient classes of deadlock-free session processes, and in shedding light on their formal underpinnings.

In session-based concurrency, types enforce correct communications through different safety and liveness properties. Basic correctness properties are *communication safety* and *session fidelity*: while the former ensures absence of errors (e.g., communication mismatches), the latter ensures that well-typed processes respect the protocols prescribed by session types. Moreover, a central (liveness) property for safe processes is that they should never "get stuck". This is the well-known *progress* property, which asserts that a well-typed term either is a final value or can further reduce [17]. In calculi for concurrency, this property has been formalized as *deadlock freedom* ("a process is deadlock-free if it can always reduce until it eventually terminates, unless the whole process diverges" [15]) or as *lock freedom* ("a process is lock free if it can always reduce until it eventually terminates, even if the whole process diverges" [13]). Notice that in the absence of divergent behaviors, deadlock and lock freedom coincide.

(Dead)lock freedom guarantees that all communications will eventually succeed, an appealing requirement for communicating processes. Several advanced type disciplines that ensure deadlock-free processes have been proposed (see, e.g., [2, 3, 5, 10, 13, 15, 16, 20]). Unfortunately, these disciplines consider different process languages and/or are based on rather different principles. As a result, very little

S. Crafa and D. Gebler (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2015)
EPTCS 190, 2015, pp. 1–15, doi:10.4204/EPTCS.190.1

© O. Dardha & J.A. Pérez
This work is licensed under the Creative Commons Attribution License.

is known about how they relate to each other. This begs several research questions: What is the formal relationship between these type disciplines? What classes of deadlock-free processes do they induce?

In this paper, we tackle these open questions by comparing $\mathscr{L}$ and $\mathscr{K}$, two salient classes of deadlock-free, session typed processes (Definition 4.2):

- $\mathscr{L}$ contains all session processes that are well-typed according to the Curry-Howard correspondence of linear logic propositions as session types [2, 3, 21]. This suffices, because the type system derived from such a correspondence ensures communication safety, session fidelity, and deadlock freedom.
- $\mathscr{K}$ contains all session processes that enjoy communication safety and session fidelity (as ensured by the type system of Vasconcelos [19]) and are (dead)lock-free by combining Kobayashi's type system based on *usages* [13, 15] with Dardha et al.'s encodability result [8].

There are good reasons for considering $\mathscr{L}$ and $\mathscr{K}$. On the one hand, due to its deep logical foundations, $\mathscr{L}$ appears to us as the *canonic* class of deadlock-free session processes, upon which all other classes should be compared. Indeed, this class arguably offers the most principled yardstick for comparisons. On the other hand, $\mathscr{K}$ integrates session type checking with the sophisticated usage discipline developed by Kobayashi for π-calculus processes. This indirect approach to deadlock freedom (first suggested in [14], later developed in [4, 7, 8]) is fairly general, as it may capture sessions with subtyping, polymorphism, and higher-order communication. Also, as informally shown in [4], $\mathscr{K}$ strictly includes classes of typed processes induced by other type systems for deadlock freedom in sessions [5, 10, 16].

One key observation in our development is that $\mathscr{K}$ corresponds to a *family* of classes of deadlock-free processes, denoted $\mathscr{K}_0, \mathscr{K}_1, \cdots, \mathscr{K}_n$, which is defined by the *degree of sharing* between their parallel components. Intuitively, $\mathscr{K}_0$ is the subclass of $\mathscr{K}$ with *independent parallel composition*: for all processes $P \mid Q \in \mathscr{K}_0$, subprocesses P and Q do not share any sessions. Then, $\mathscr{K}_1$ is the subclass of $\mathscr{K}$ which contains $\mathscr{K}_0$ but admits also processes with parallel components that share at most one session. Then, $\mathscr{K}_n$ contains deadlock-free session processes whose parallel components share at most n sessions.

Contributions. In this paper, we present three main contributions:

1. We show that the inclusion between the constituent classes of $\mathscr{K}$ is *strict* (Theorem 4.4). We have:

$$\mathscr{K}_0 \subset \mathscr{K}_1 \subset \mathscr{K}_2 \subset \cdots \subset \mathscr{K}_n \subset \mathscr{K}_{n+1} \tag{1}$$

   Although not extremely surprising, the significance of this result lies in the fact that it talks about concurrency (via the degree of sharing) but implicitly also about the potential sequentiality of parallel processes. As such, processes in $\mathscr{K}_k$ are necessarily "more parallel" than those in $\mathscr{K}_{k+1}$. Interestingly, the degree of sharing in $\mathscr{K}_0, \ldots, \mathscr{K}_n$ can be defined in a very simple way, via a natural condition in the rule for parallel composition in Kobayashi's type system for deadlock freedom.

2. We show that $\mathscr{L}$ and $\mathscr{K}_1$ coincide (Theorem 4.6). That is, there are deadlock-free session processes that cannot be typed by systems derived from the Curry-Howard interpretation of session types [2, 3, 21], but that can be admitted by the (indirect) approach of [8]. This result is significant: it establishes the precise status of systems based on [3,21] with respect to previous (non Curry-Howard) disciplines. Indeed, it formally confirms that linear logic interpretations of session types naturally induce the most basic form of concurrent cooperation (sharing of exactly one session), embodied as the principle of "composition plus hiding", a distinguishing feature of such interpretations.

3. We define a rewriting procedure of processes in $\mathscr{K}$ into $\mathscr{L}$ (Defintion 5.7). Intuitively, due to our previous observation and characterization of the degree of sharing in session typed processes, it is

quite natural to convert a process in $\mathscr{K}$ into another, more parallel process in $\mathscr{L}$. In essence, the procedure replaces sequential prefixes with representative parallel components. The rewriting procedure satisfies type-preservation, and enjoys the compositionality and operational correspondence criteria as stated in [11] (cf. Theorems 5.8 and 5.10). These properties not only witness the significance of the rewriting procedure; they also confirm that the degree of sharing is a rather subtle criteria for formally distinguishing deadlock-free, session typed processes.

To the best of our knowledge, these contributions define the first formal comparison between fundamentally distinct type systems for deadlock freedom in session communications. Previous comparisons, such as the ones in [4] and [3, §6], are informal: they are based on representative "corner cases", i.e., examples of deadlock-free session processes typable in one system but not in some other.

The paper is structured as follows. § 2 summarizes the session π-calculus and associated type system of [19]. In § 3 we present the two typed approaches to deadlock freedom for sessions. § 4 defines the classes $\mathscr{L}$ and $\mathscr{K}$, formalizes the hierarchy (1), and shows that $\mathscr{L}$ and $\mathscr{K}_1$ coincide. In § 5 we give the rewriting procedure of $\mathscr{K}_n$ into $\mathscr{L}$ and establish its properties. § 6 collects some concluding remarks. Due to space restrictions, details of proofs are omitted; they can be found online [9].

2 Session π-calculus

Following Vasconcelos [19], we introduce the session π-calculus and its associated type system which ensures communication safety and session fidelity. The syntax is given in Figure 1 (upper part). Let P, Q range over processes x, y over channels and v over values; for simplicity, the set of values coincides with that of channels. In examples, we often use **n** to denote a terminated channel that cannot be further used.

Process $\overline{x}\langle v\rangle.P$ denotes the output of v along x, with continuation P. Dually, process $x(y).P$ denotes an input along x with continuation P, with y denoting a placeholder. Process $x \triangleleft l_j.P$ uses x to select l_j from a labelled choice process, being $x \triangleright \{l_i : P_i\}_{i\in I}$, so as to trigger P_j; labels indexed by the finite set I are pairwise distinct. We also have the inactive process (denoted **0**), the parallel composition of P and Q (denoted $P \mid Q$), and the (double) restriction operator, noted $(\nu xy)P$: the intention is that x and y denote *dual session endpoints* in P. We omit **0** whenever possible and write, e.g., $\overline{x}\langle \mathbf{n}\rangle$ instead of $\overline{x}\langle \mathbf{n}\rangle.\mathbf{0}$. Notions of bound/free variables in processes are standard; we write $\mathsf{fn}(P)$ to denote the set of free names of P. Also, we write $P[v/z]$ to denote the (capture-avoiding) substitution of free occurrences of z in P with v.

The operational semantics is given in terms of a reduction relation, noted $P \to Q$, and defined by the rules in Figure 1 (lower part). It relies on a standard notion of structural congruence, noted $\equiv$ (see [19]). We write $\to^*$ to denote the reflexive, transitive closure of $\to$. Observe that interaction involves prefixes with different channels (endpoints), and always occurs in the context of an outermost (double) restriction. Key rules are (R-Com) and (R-Case), denoting the interaction of output/input prefixes and selection/branching constructs, respectively. Rules (R-Par), (R-Res), and (R-Str) are standard.

The syntax of session types, ranged over $T, S, \ldots$, is given by the following grammar.

$$T, S ::= \mathbf{end} \;\mid\; ?T.S \;\mid\; !T.S \;\mid\; \&\{l_i : S_i\}_{i\in I} \;\mid\; \oplus\{l_i : S_i\}_{i\in I}$$

Above, **end** is the type of an endpoint with a terminated protocol. The type $?T.S$ is assigned to an endpoint that first receives a value of type T and then continues according to the protocol described by S. Dually, type $!T.S$ is assigned to an endpoint that first outputs a value of type T and then continues according to the protocol described by S. Type $\oplus\{l_i : S_i\}_{i\in I}$, an *internal choice*, generalizes output types; type $\&\{l_i : S_i\}_{i\in I}$, an *external choice*, generalizes input types. Notice that session types describe *sequences* of structured behaviors; they do not admit parallel composition operators.

$$
\begin{array}{rlll}
P,Q ::= & \overline{x}\langle v\rangle.P & \text{(output)} & \mathbf{0} \quad \text{(inaction)} \\
& x(y).P & \text{(input)} & P \mid Q \quad \text{(composition)} \\
& x \triangleleft l_j.P & \text{(selection)} & (\nu xy)P \quad \text{(session restriction)} \\
& x \triangleright \{l_i : P_i\}_{i\in I} & \text{(branching)} & \\
v ::= & x & \text{(channel)} &
\end{array}
$$

$$
\begin{array}{ll}
\text{(R-Com)} & (\nu xy)(\overline{x}\langle v\rangle.P \mid y(z).Q) \to (\nu xy)(P \mid Q[v/z]) \\
\text{(R-Case)} & (\nu xy)(x \triangleleft l_j.P \mid y \triangleright \{l_i : P_i\}_{i\in I}) \to (\nu xy)(P \mid P_j) \quad j \in I \\
\text{(R-Str)} & P \equiv P',\ P \to Q,\ Q' \equiv Q \Longrightarrow P' \to Q'
\end{array}
\qquad
\begin{array}{ll}
\text{(R-Par)} & P \to Q \Longrightarrow P \mid R \to Q \mid R \\
\text{(R-Res)} & P \to Q \Longrightarrow (\nu xy)P \to (\nu xy)Q
\end{array}
$$

Figure 1: Session π-calculus: syntax and semantics.

$$
\text{(T-Nil)}\ \frac{}{x : \mathbf{end} \vdash_{\mathrm{ST}} \mathbf{0}}
\qquad
\text{(T-Par)}\ \frac{\Gamma_1 \vdash_{\mathrm{ST}} P \quad \Gamma_2 \vdash_{\mathrm{ST}} Q}{\Gamma_1 \circ \Gamma_2 \vdash_{\mathrm{ST}} P \mid Q}
\qquad
\text{(T-Res)}\ \frac{\Gamma, x : T, y : \overline{T} \vdash_{\mathrm{ST}} P}{\Gamma \vdash_{\mathrm{ST}} (\nu xy)P}
\qquad
\text{(T-In)}\ \frac{\Gamma, x : S, y : T \vdash_{\mathrm{ST}} P}{\Gamma, x : ?T.S \vdash_{\mathrm{ST}} x(y).P}
$$

$$
\text{(T-Out)}\ \frac{\Gamma, x : S \vdash_{\mathrm{ST}} P}{\Gamma, x : !T.S, y : T \vdash_{\mathrm{ST}} \overline{x}\langle y\rangle.P}
\qquad
\text{(T-Brch)}\ \frac{\Gamma, x : S_i \vdash_{\mathrm{ST}} P_i \quad \forall i \in I}{\Gamma, x : \&\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleright \{l_i : P_i\}_{i\in I}}
\qquad
\text{(T-Sel)}\ \frac{\Gamma, x : S_j \vdash_{\mathrm{ST}} P \quad \exists j \in I}{\Gamma, x : \oplus\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P}
$$

Figure 2: Typing rules for the π-calculus with sessions.

A central notion in session-based concurrency is *duality*, which relates session types offering opposite (i.e., complementary) behaviors. Duality stands at the basis of communication safety and session fidelity. Given a session type T, its dual type $\overline{T}$ is defined as follows:

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
\overline{!T.S} & \triangleq & ?T.\overline{S} & \overline{?T.S} & \triangleq & !T.\overline{S} & \overline{\mathbf{end}} & \triangleq & \mathbf{end} \\
\overline{\oplus\{l_i : S_i\}_{i\in I}} & \triangleq & \&\{l_i : \overline{S}_i\}_{i\in I} & \overline{\&\{l_i : S_i\}_{i\in I}} & \triangleq & \oplus\{l_i : \overline{S}_i\}_{i\in I} & & &
\end{array}
$$

Typing contexts, ranged over by Γ, Γ', are sets of typing assignments $x : T$. Given a context Γ and a process P, a session typing judgement is of the form $\Gamma \vdash_{\mathrm{ST}} P$. Typing rules are given in Figure 2. Rule (T-Nil) states that $\mathbf{0}$ is well-typed under a terminated channel. Rule (T-Par) types the parallel composition of two processes by composing their corresponding typing contexts using a splitting operator, noted $\circ$ [19]. Rule (T-Res) types a restricted process by requiring that the two endpoints have dual types. Rules (T-In) and (T-Out) type the receiving and sending of a value over a channel x, respectively. Finally, rules (T-Brch) and (T-Sel) are generalizations of input and output over a labelled set of processes.

The main guarantees of the type system are *communication safety* and *session fidelity*, i.e., typed processes respect their ascribed protocols, as represented by session types.

Theorem 2.1 (Type Preservation for Session Types)**.** *If $\Gamma \vdash_{\mathrm{ST}} P$ and $P \to Q$, then $\Gamma \vdash_{\mathrm{ST}} Q$.*

The following notion of well-formed processes is key to single out meaningful typed processes.

Definition 2.2 (Well-Formedness for Sessions)**.** *A process is* well-formed *if for any of its structural congruent processes of the form $(\nu\widetilde{xy})(P \mid Q)$ the following hold.*

- *If P and Q are prefixed at the same variable, then the variable performs the same action (input or output, branching or selection).*

- *If P is prefixed in x_i and Q is prefixed in y_i where $x_i y_i \in \widetilde{xy}$, then $P \mid Q \rightarrow$.*

It is important to notice that well-typedness of a process does not imply the process is well-formed. We have the following theorem:

Theorem 2.3 (Type Safety for Sessions [19])**.** *If $\vdash_{\mathtt{ST}} P$ then P is well-formed.*

We present the main result of the session type system. The following theorem states that a well-typed closed process does not reduce to an ill-formed one. It follows immediately from Theorems 2.1 and 2.3.

Theorem 2.4 ([19])**.** *If $\vdash_{\mathtt{ST}} P$ and $P \rightarrow^* Q$, then Q is well-formed.*

An important observation is that the session type system given above does not exclude *deadlocked processes*, i.e., processes which reach a "stuck state." This is because the interleaving of communication prefixes in typed processes may create extra causal dependencies not described by session types. (This intuitive definition of deadlocked processes will be made precise below.) A particularly insidious class of deadlocks is due to cyclic interleaving of channels in processes. For example, consider a process such as $P \triangleq (\nu xy)(\nu wz)(\overline{x}\langle \mathbf{n} \rangle.\overline{w}\langle \mathbf{n} \rangle \mid z(t).y(s))$: it represents the implementation of two (simple) independent sessions, which get intertwined (blocked) due to the nesting induced by input and output prefixes. We have that $\mathbf{n} : \mathbf{end} \vdash_{\mathtt{ST}} P$ even if P is unable to reduce. A deadlock-free variant of P would be, e.g., process $P' \triangleq (\nu xy)(\nu wz)(\overline{x}\langle \mathbf{n} \rangle.\overline{w}\langle \mathbf{n} \rangle \mid y(s).z(t))$, which also is typable in $\vdash_{\mathtt{ST}}$.

We will say that a process is *deadlock-free* if any communication action that becomes active during execution is eventually consumed; that is, there is a corresponding co-action that eventually becomes available. Below we define deadlock freedom in the session π-calculus; we follow [13, 15] and consider *fair* reduction sequences [6]. For simplicity, we omit the symmetric cases for input and branching.

Definition 2.5 (Deadlock Freedom for Session π-Calculus)**.** *A process P_0 is* deadlock-free *if for any fair reduction sequence $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots$, we have that*

1. *$P_i \equiv (\nu \widetilde{xy})(\overline{x}\langle v \rangle.Q \mid R)$, for $i \geq 0$, implies that there exists $n \geq i$ such that $P_n \equiv (\nu \widetilde{x'y'})(\overline{x}\langle v \rangle.Q \mid y(z).R_1 \mid R_2)$ and $P_{n+1} \equiv (\nu \widetilde{x'y'})(Q \mid R_1[v/z] \mid R_2)$;*
2. *$P_i \equiv (\nu \widetilde{xy})(x \triangleleft l_j.Q \mid R)$, for $i \geq 0$, implies that there exists $n \geq i$ such that $P_n \equiv (\nu \widetilde{x'y'})(x \triangleleft l_j.Q \mid y \triangleright \{l_k : R_k\}_{k \in I \cup \{j\}} \mid S)$ and $P_{n+1} \equiv (\nu \widetilde{x'y'})(Q \mid R_j \mid S)$.*

3 Two Approaches to Deadlock Freedom

We introduce two approaches to deadlock-free, session typed processes. The first one, given in § 3.1, comes from interpretations of linear logic propositions as session types [1–3, 21]; the second approach, summarized in § 3.2, combines usage types for the standard π-calculus with encodings of session processes and types [8]. Based on these two approaches, in § 4 we will define the classes $\mathscr{L}$ and $\mathscr{K}$.

3.1 Linear Logic Foundations of Session Types

The linear logic interpretation of session types was introduced by Caires and Pfenning [3], and developed by Wadler [21] and others. Initially proposed for intutitionistic linear logic, here we consider an interpretation based on classical linear logic with mix principles, following a recent presentation by Caires [1].

The syntax and semantics of processes are as in § 2 except for the following differences. First, we have the standard restriction construct $(\nu x)P$, which replaces the double restriction. Second, we have a so-called *forwarding process*, denoted $[x \leftrightarrow y]$, which intuitively "fuses" names x and y. Besides these

$$\text{(T-}\mathbf{1}\text{)}\ \frac{}{\mathbf{0}\vdash_{\mathtt{CH}} x{:}\bullet} \qquad \text{(T-}\bot\text{)}\ \frac{P\vdash_{\mathtt{CH}}\Delta}{P\vdash_{\mathtt{CH}} x{:}\bullet,\Delta} \qquad \text{(T-id)}\ \frac{}{[x\leftrightarrow y]\vdash_{\mathtt{CH}} x{:}A, y{:}\overline{A}}$$

$$\text{(T-}\parr\text{)}\ \frac{P\vdash_{\mathtt{CH}}\Delta, y{:}A, x{:}B}{x(y).P\vdash_{\mathtt{CH}}\Delta, x{:}A\parr B} \qquad \text{(T-}\otimes\text{)}\ \frac{P\vdash_{\mathtt{CH}}\Delta, y{:}A \qquad Q\vdash_{\mathtt{CH}}\Delta', x{:}B}{\overline{x}(y).(P\mid Q)\vdash_{\mathtt{CH}}\Delta,\Delta', x{:}A\otimes B} \qquad \text{(T-cut)}\ \frac{P\vdash_{\mathtt{CH}}\Delta, x{:}\overline{A} \qquad Q\vdash_{\mathtt{CH}}\Delta', x{:}A}{(\nu x)(P\mid Q)\vdash_{\mathtt{CH}}\Delta,\Delta'}$$

$$\text{(T-}\oplus\text{)}\ \frac{P\vdash_{\mathtt{CH}}\Delta, x{:}A_j \qquad j\in I}{x\triangleleft l_j.P\vdash_{\mathtt{CH}}\Delta, x{:}\oplus\{l_i:A_i\}_{i\in I}} \qquad \text{(T-\&)}\ \frac{P_i\vdash_{\mathtt{CH}}\Delta, x{:}A_i \qquad \forall i\in I}{x\triangleright\{l_i:P_i\}_{i\in I}\vdash_{\mathtt{CH}}\Delta, x{:}\&\{l_i:A_i\}_{i\in I}} \qquad \text{(T-mix)}\ \frac{P\vdash_{\mathtt{CH}}\Delta \qquad Q\vdash_{\mathtt{CH}}\Delta'}{P\mid Q\vdash_{\mathtt{CH}}\Delta,\Delta'}$$

Figure 3: Typing rules for the π-calculus with C-types.

differences in syntax, we have also some minor modifications in reduction rules. Differences with respect to the language considered in § 2 are summarized in the following:

$$P,Q ::= \quad (\nu x)P \quad \text{(channel restriction)} \quad \mid \quad [x\leftrightarrow y] \quad \text{(forwarding)}$$

$$\begin{array}{llll} \text{(R-ChCom)} & \overline{x}\langle v\rangle.P\mid x(z).Q\to P\mid Q[v/z] & \text{(R-Fwd)} & (\nu x)([x\leftrightarrow y]\mid P)\to P[y/x] \\ \text{(R-ChCase)} & x\triangleleft l_j.P\mid x\triangleright\{l_i:P_i\}_{i\in I}\to P\mid P_j \quad j\in I & \text{(R-ChRes)} & P\to Q\Longrightarrow(\nu x)P\to(\nu x)Q \end{array}$$

Observe how interaction of input/output prefixes and selection/branching is no longer covered by an outermost restriction. As for the type system, we consider the so-called C-types which correspond to linear logic propositions. They are given by the following grammar:

$$A,B ::= \bot \ \mid\ \mathbf{1} \ \mid\ A\otimes B \ \mid\ A\parr B \ \mid\ \oplus\{l_i:A_i\}_{i\in I} \ \mid\ \&\{l_i:A_i\}_{i\in I}$$

Intuitively, $\bot$ and $\mathbf{1}$ are used to type a terminated endpoint. Type $A\otimes B$ is associated to an endpoint that first outputs an object of type A and then behaves according to B. Dually, type $A\parr B$ is the type of an endpoint that first inputs an object of type A and then continues as B. The interpretation of $\oplus\{l_i:A_i\}_{i\in I}$ and $\&\{l_i:A_i\}_{i\in I}$ as select and branch behaviors follows as expected.

We define a full duality on C-types, which exactly corresponds to the negation operator of CLL $(\cdot)^{\bot}$. The *dual* of type A, denoted $\overline{A}$, is inductively defined as follows:

$$\begin{array}{rclcrclcrcl} \overline{\mathbf{1}} & = & \bot & \qquad & \overline{\bot} & = & \mathbf{1} & \qquad & \overline{\oplus\{l_i:A_i\}_{i\in I}} & = & \&\{l_i:\overline{A}_i\}_{i\in I} \\ \overline{A\otimes B} & = & \overline{A}\parr\overline{B} & \qquad & \overline{A\parr B} & = & \overline{A}\otimes\overline{B} & \qquad & \overline{\&\{l_i:A_i\}_{i\in I}} & = & \oplus\{l_i:\overline{A}_i\}_{i\in I} \end{array}$$

Recall that $A\multimap B \triangleq \overline{A}\parr B$. As explained in [1], considering mix principles means admitting $\bot\multimap\mathbf{1}$ and $\mathbf{1}\multimap\bot$, and therefore $\bot=\mathbf{1}$. We write $\bullet$ to denote either $\bot$ or $\mathbf{1}$, and decree that $\overline{\bullet}=\bullet$.

Typing contexts, sets of typing assignments $x:A$, are ranged over $\Delta,\Delta',\ldots$. The empty context is denoted '$\cdot$'. Typing judgments are then of the form $P\vdash_{\mathtt{CH}}\Delta$. Figure 3 gives the typing rules associated to the linear logic interpretation. Salient points include the use of bound output $(\nu y)\overline{x}\langle y\rangle.P$, which is abbreviated as $\overline{x}(y)P$. Another highlight is the "composition plus hiding" principle implemented by rule (T-cut), which integrates parallel composition and restriction in a single rule. Indeed, there is no dedicated rule for restriction. Also, rule (T-mix) enables the typing of *independent parallel compositions*, i.e., the composition of two processes that do not share sessions.

We now collect main results for this type system; see [1, 3] for details. For any P, define $live(P)$ if and only if $P\equiv(\nu\widetilde{n})(\pi.Q\mid R)$, where π is an input, output, selection, or branching prefix.

$$
\begin{array}{rlll}
U ::= & ?^{o}_{\kappa}.U & \text{(used in input)} & \emptyset \quad \text{(not usable)} \\
 & !^{o}_{\kappa}.U & \text{(used in output)} & (U_1 \mid U_2) \quad \text{(used in parallel)} \\
T ::= & U[\widetilde{T}] & \text{(channel types)} & \langle l : T \rangle_{i \in I} \quad \text{(variant type)}
\end{array}
$$

Figure 4: Syntax of usage types for the π-calculus.

Theorem 3.1 (Type Preservation for C-Types)**.** *If $P \vdash_{\mathtt{CH}} \Delta$ and $P \longrightarrow Q$ then $Q \vdash_{\mathtt{CH}} \Delta$.*

Theorem 3.2 (Progress)**.** *If $P \vdash_{\mathtt{CH}} \cdot$ and $live(P)$ then $P \longrightarrow Q$, for some Q.*

3.2 Deadlock Freedom by Encodability

As mentioned above, the second approach to deadlock-free session processes is *indirect*, in the sense that establishing deadlock freedom for session processes appeals to usage types for the π-calculus [13,15], for which type systems enforcing deadlock freedom are well-established. Formally, this reduction exploits encodings of processes and types: a session process $\Gamma \vdash_{\mathtt{ST}} P$ is encoded into a (standard) π-calculus process $[\![\Gamma]\!]_f \vdash^{n}_{\mathtt{KB}} [\![P]\!]_f$. Next we introduce the syntax of standard π-calculus processes with variant values (§ 3.2.1), the discipline of usage types (§ 3.2.2), and the encodings of session processes and types into standard π-calculus processes and usage types, respectively (§ 3.2.3).

3.2.1 Processes

The syntax and semantics of the π-calculus with usage types build upon those in § 2. We require some modifications. First, the encoding of terms presented in § 3.2.3, requires polyadic communication. Rather than branching and selection constructs, the π-calculus that we consider here includes a *case* construct $\mathbf{case}\, v\, \mathbf{of}\, \{l_i\_x_i \triangleright P_i\}_{i \in I}$ that uses *variant value* l_j_v. Moreover, we consider the standard channel restriction, rather than double restriction. These modifications are summarized below:

$$
\begin{array}{rlll}
P, Q ::= & (\nu x)P & \text{(channel restriction)} \quad \mid & \mathbf{case}\, v\, \mathbf{of}\, \{l_i\_x_i \triangleright P_i\}_{i \in I} \quad \text{(case)} \\
v ::= & l_j\_v & \text{(variant value)} &
\end{array}
$$

$$
\begin{array}{ll}
(\mathrm{R}\pi\text{-}\textsc{Com}) & \overline{x}\langle \widetilde{v} \rangle.P \mid x(\widetilde{z}).Q \rightarrow P \mid Q[\widetilde{v}/\widetilde{z}] \\
(\mathrm{R}\pi\text{-}\textsc{Res}) & P \rightarrow Q \Longrightarrow (\nu x)P \rightarrow (\nu x)Q \\
(\mathrm{R}\pi\text{-}\textsc{Case}) & \mathbf{case}\, l_j\_v\, \mathbf{of}\, \{l_i\_x_i \triangleright P_i\}_{i \in I} \rightarrow P_j[v/x_i] \quad j \in I
\end{array}
$$

The definition of deadlock-freedom for the π-calculus follows [13, 15]:

Definition 3.3 (Deadlock Freedom for Standard π-Calculus)**.** *A process P_0 is* deadlock-free *under fair scheduling, if for any fair reduction sequence $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \cdots$ the following hold*

1. *if $P_i \equiv (\nu \tilde{x})(\overline{x}\langle \widetilde{v} \rangle.Q \mid R)$ for $i \geq 0$, implies that there exists $n \geq i$ such that $P_n \equiv (\nu \tilde{x})(\overline{x}\langle \widetilde{v} \rangle.Q \mid x(\widetilde{z}).R_1 \mid R_2)$ and $P_{n+1} \equiv (\nu \tilde{x})(Q \mid R_1[\widetilde{v}/\widetilde{z}] \mid R_2)$;*
2. *if $P_i \equiv (\nu \tilde{x})(x(\widetilde{z}).Q \mid R)$ for $i \geq 0$, implies that there exists $n \geq i$ such that $P_n \equiv (\nu \tilde{x})(x(\widetilde{z}).Q \mid \overline{x}\langle \widetilde{v} \rangle.R_1 \mid R_2)$ and $P_{n+1} \equiv (\nu \tilde{x})(Q[\widetilde{v}/\widetilde{z}] \mid R_1 \mid R_2)$.*

3.2.2 Usage Types

The syntax of usage types is defined in Figure 4. For simplicity, we let α range over input ? or output ! actions. The usage $\emptyset$ describes a channel that cannot be used at all. We will often omit $\emptyset$, and so we

will write U instead of $U.\emptyset$. Usages $?^{o}_{\kappa}.U$ and $!^{o}_{\kappa}.U$ describe channels that can be used once for input and output, respectively and then used according to the continuation usage U. The *obligation* o and *capability* κ range over the set of natural numbers. The usage $U_1 \mid U_2$ describes a channel that is used according to U_1 by one process and U_2 by another processes in parallel.

Intuitively, obligations and capabilities describe inter-channel dependencies:

- An obligation of level n must be fulfilled by using only capabilities of level *less than* n. Said differently, an action of obligation n must be prefixed by actions of capabilities less than n.
- For an action with capability of level n, there must exist a co-action with obligation of level *less than or equal to* n.

Typing contexts are sets of typing assignments and are ranged over Γ, Γ'. A typing judgement is of the form $\Gamma \vdash^{n}_{\mathrm{KB}} P$: the annotation n explicitly denotes the greatest *degree of sharing* admitted in parallel processes. Before commenting on the typing rules (given in Figure 5), we discuss some important auxiliary notions, extracted from [13, 15]. First, the composition operation on types (denoted $\mid$, and used in rules Tπ-(PAR)$_n$ and Tπ-(OUT)) is based on the composition of usages and is defined as follows:

$$\langle l_i : T_i \rangle_{i \in I} \mid \langle l_i : T_i \rangle_{i \in I} = \langle l_i : T_i \rangle_{i \in I} \qquad U_1[\widetilde{T}] \mid U_2[\widetilde{T}] = (U_1 \mid U_2)[\widetilde{T}]$$

The generalization of $\mid$ to typing contexts, denoted $(\Gamma_1 \mid \Gamma_2)(x)$, is defined as expected. The unary operation $\uparrow^t$ applied to a usage U lifts its obligation level *up to* t; it is defined inductively as:

$$\uparrow^t \emptyset = \emptyset \qquad \uparrow^t \alpha^{o}_{\kappa}.U = \alpha^{max(o,t)}_{\kappa}.U \qquad \uparrow^t (U_1 \mid U_2) = (\uparrow^t U_1 \mid \uparrow^t U_2)$$

The $\uparrow^t$ is extends to types/typing contexts as expected. *Duality* on usage types simply exchanges ? and !:

$$\overline{\emptyset[]} = \emptyset[] \qquad \overline{?^{o}_{\kappa}.U[\widetilde{T}]} = !^{o}_{\kappa}.\overline{U}[\widetilde{T}] \qquad \overline{!^{o}_{\kappa}.U[\widetilde{T}]} = ?^{o}_{\kappa}.\overline{U}[\widetilde{T}]$$

Operator " ; " in $\Delta = x : [T]\alpha^{o}_{\kappa}$; Γ, used in rules (Tπ-IN) and (Tπ-OUT), is such that the following hold:

$$\mathsf{dom}(\Delta) = \{x\} \cup \mathsf{dom}(\Gamma) \qquad \Delta(x) = \begin{cases} \alpha^{o}_{\kappa}.U[\widetilde{T}] & \text{if } \Gamma(x) = U[\widetilde{T}] \\ \alpha^{o}_{\kappa}[\widetilde{T}] & \text{if } x \notin dom(\Gamma) \end{cases} \qquad \Delta(y) = \uparrow^{\kappa+1} \Gamma(y) \text{ if } y \neq x$$

The final required notion is that of a *reliable usage*. It builds upon the following definition:

Definition 3.4. *Let U be a usage. The input and output* obligation levels *(resp.* capability levels*) of U, written $\mathsf{ob}_?(U)$ and $\mathsf{ob}_!(U)$ (resp. $\mathsf{cap}_?(U)$ and $\mathsf{cap}_!(U)$), are defined as:*

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{ob}_\alpha(\alpha^{o}_{\kappa}.U) & = & o & \mathsf{cap}_\alpha(\alpha^{o}_{\kappa}.U) & = & \kappa \\ \mathsf{ob}_\alpha(U_1 \mid U_2) & = & min(\mathsf{ob}_\alpha(U_1), \mathsf{ob}_\alpha(U_2)) & \mathsf{cap}_\alpha(U_1 \mid U_2) & = & min(\mathsf{cap}_\alpha(U_1), \mathsf{cap}_\alpha(U_2)) \end{array}$$

The definition of reliable usages depends on a reduction relation on usages, noted $U \to U'$. Intuitively, $U \to U'$ means that if a channel of usage U is used for communication, then after the communication occurs, the channel should be used according to usage U'. Thus, e.g., $?^{o}_{\kappa}.U_1 \mid ?^{o'}_{\kappa'}.U_2$ reduces to $U_1 \mid U_2$.

Definition 3.5 (Reliability)**.** *We write $\mathsf{con}_\alpha(U)$ when $\mathsf{ob}_{\overline{\alpha}}(U) \leq \mathsf{cap}_\alpha(U)$. We write $\mathsf{con}(U)$ when $\mathsf{con}_?(U)$ and $\mathsf{con}_!(U)$ hold. Usage U is* reliable*, noted $\mathsf{rel}(U)$, if $\mathsf{con}(U')$ holds $\forall U'$ such that $U \to^* U'$.*

$$\frac{}{x:\emptyset[] \vdash^n_{\mathtt{KB}} \mathbf{0}}\ (\text{T}\pi\text{-Nil}) \qquad \frac{\Gamma, x:U[\widetilde{T}] \vdash^n_{\mathtt{KB}} P \quad \mathsf{rel}(U)}{\Gamma \vdash^n_{\mathtt{KB}} (\nu x)P}\ (\text{T}\pi\text{-Res}) \qquad \frac{\Gamma_1 \vdash^n_{\mathtt{KB}} P \qquad \Gamma_2 \vdash^n_{\mathtt{KB}} Q \qquad |\Gamma_1 \cap \Gamma_1| \leq n}{\Gamma_1 \mid \Gamma_2 \vdash^n_{\mathtt{KB}} P \mid Q}\ (\text{T}\pi\text{-Par}_n)$$

$$\frac{\Gamma, \widetilde{y}:\widetilde{T} \vdash^n_{\mathtt{KB}} P}{x:?^0_\kappa[\widetilde{T}]\ ;\ \Gamma \vdash^n_{\mathtt{KB}} x(\widetilde{y}).P}\ (\text{T}\pi\text{-In})$$

$$\frac{\Gamma_1 \vdash^n_{\mathtt{KB}} \tilde{v}:\widetilde{T} \qquad \Gamma_2 \vdash^n_{\mathtt{KB}} P}{x:!^0_\kappa[\widetilde{T}]\ ;\ (\Gamma_1 \mid \Gamma_2) \vdash^n_{\mathtt{KB}} \overline{x}\langle\tilde{v}\rangle.P}\ (\text{T}\pi\text{-Out}) \qquad \frac{\Gamma \vdash^n_{\mathtt{KB}} v:T_j \quad \exists j \in I}{\Gamma \vdash^n_{\mathtt{KB}} l_j\_v : \langle l_i:T_i\rangle_{i\in I}}\ (\text{T}\pi\text{-LVal})$$

$$\frac{\Gamma_1 \vdash^n_{\mathtt{KB}} v:\langle l_i:T_i\rangle_{i\in I} \qquad \Gamma_2, x_i:T_i \vdash^n_{\mathtt{KB}} P_i \quad \forall i \in I}{\Gamma_1,\Gamma_2 \vdash^n_{\mathtt{KB}} \mathbf{case}\, v\, \mathbf{of}\, \{l_i\_x_i \triangleright P_i\}_{i\in I}}\ (\text{T}\pi\text{-Case})$$

Figure 5: Typing rules for the π-calculus with usage types with degree of sharing n.

Typing Rules. The typing rules for the standard π-calculus with usage types are given in Figure 5. The only difference with respect to the rules in Kobayashi's systems [13, 15] is that we annotate typing judgements with the degree of sharing, explicitly stated in rule ($\text{T}\pi\text{-Par}_n$)–see below. Rule ($\text{T}\pi\text{-Nil}$) states that the terminated process is typed under a terminated channel. Rule ($\text{T}\pi\text{-Res}$) states that process $(\nu x)P$ is well-typed if the usage for x is reliable (cf. Definition 3.5). Rules ($\text{T}\pi\text{-In}$) and ($\text{T}\pi\text{-Out}$) type input and output processes in a typing context where the " ; " operator is used in order to increase the obligation level of the channels in continuation P. Rules ($\text{T}\pi\text{-LVal}$) and ($\text{T}\pi\text{-Case}$) type a choice: the first types a variant value with a variant type; the second types a case process using a variant value as its guard.

Given a degree of sharing n, rule ($\text{T}\pi\text{-Par}_n$) states that the parallel composition of processes P and Q (typable under contexts Γ_1 and Γ_2, respectively) is well-typed under the typing context $\Gamma_1 \mid \Gamma_2$ only if $|\Gamma_1 \cap \Gamma_2| \leq n$. This allows to simply characterize the "concurrent cooperation" between P and Q. As a consequence, if $P \vdash^n_{\mathtt{KB}}$ then $P \vdash^k_{\mathtt{KB}}$, for any $k \leq n$. Observe that the typing rule for parallel composition in [13, 15] is the same as ($\text{T}\pi\text{-Par}_n$), except for condition $|\Gamma_1 \cap \Gamma_2| \leq n$, which is not specified.

The next theorems imply that well-typed processes by the type system in Figure 5 are deadlock-free.

Theorem 3.6 (Type Preservation for Usage Types)**.** *If $\Gamma \vdash^n_{\mathtt{KB}} P$ and $P \to Q$, then $\Gamma' \vdash^n_{\mathtt{KB}} Q$ for some Γ' such that $\Gamma \to \Gamma'$.*

Theorem 3.7 (Deadlock Freedom)**.** *If $\emptyset \vdash^n_{\mathtt{KB}} P$ and either $P \equiv (\nu\tilde{x})(x(\widetilde{z}).Q \mid R)$ or $P \equiv (\nu\tilde{x})(\overline{x}\langle\widetilde{v}\rangle.Q \mid R)$, then $P \to Q$, for some Q.*

Corollary 3.8. *If $\emptyset \vdash^n_{\mathtt{KB}} P$, then P is deadlock-free, in the sense of Definition 3.3.*

Theorem 3.2 (progress for the linear logic system) and Theorem 3.7 (deadlock freedom for the standard π-calculus) have a rather similar formulation: both properties state that processes can always reduce if they are well-typed (under the empty typing context) and have an appropriate structure (i.e., condition $live(P)$ in Theorem 3.2 and condition $P \equiv (\nu\tilde{x})(x(\widetilde{z}).Q \mid R)$ or $P \equiv (\nu\tilde{x})(\overline{x}\langle\widetilde{v}\rangle.Q \mid R)$ in Theorem 3.7).

3.2.3 Encodings of Processes and Types

Encoding of Processes. To relate classes of processes obtained by the different type systems given so far, we rewrite a session typed or C-typed process into a usage typed process by following a continuation-passing style: this allows us to mimic the structure of a session or C-type by sending its continuation as a payload over a channel. This idea, suggested in [14] and developed in [8], is recalled in Figure 6.

$$
\begin{array}{lcllcl}
[\![\overline{x}\langle v\rangle.P]\!]_f & \triangleq & (\nu c)\overline{f_x}\langle v,c\rangle.[\![P]\!]_{f,\{x\mapsto c\}} & [\![x\triangleright\{l_i:P_i\}_{i\in I}]\!]_f & \triangleq & f_x(y).\ \mathbf{case}\, y\, \mathbf{of}\, \{l_i\_c\triangleright [\![P_i]\!]_{f,\{x\mapsto c\}}\}_{i\in I}\\
[\![x(y).P]\!]_f & \triangleq & f_x(y,c).[\![P]\!]_{f,\{x\mapsto c\}} & [\![(\nu xy)P]\!]_f & \triangleq & (\nu c)[\![P]\!]_{f,\{x,y\mapsto c\}}\\
[\![x\triangleleft l_j.P]\!]_f & \triangleq & (\nu c)\overline{f_x}\langle l_j\_c\rangle.[\![P]\!]_{f,\{x\mapsto c\}} & [\![P\mid Q]\!]_f & \triangleq & [\![P]\!]_f\mid[\![Q]\!]_f
\end{array}
$$

Figure 6: Encoding of session processes into π-calculus processes.

$$
\begin{array}{rcl@{\qquad}rcl}
[\![\mathbf{end}]\!]_{\mathsf{su}} & = & \emptyset[] & [\![\mathbf{end}]\!]_{\mathsf{c}} & = & \bullet\\
[\![?T.S]\!]_{\mathsf{su}} & = & ?^{\mathsf{o}}_{\kappa}[[\![T]\!]_{\mathsf{su}},[\![S]\!]_{\mathsf{su}}] & [\![?T.S]\!]_{\mathsf{c}} & = & [\![T]\!]_{\mathsf{c}}\parr[\![S]\!]_{\mathsf{c}}\\
[\![!T.S]\!]_{\mathsf{su}} & = & !^{\mathsf{o}}_{\kappa}[[\![T]\!]_{\mathsf{su}},[\![\overline{S}]\!]_{\mathsf{su}}] & [\![!T.S]\!]_{\mathsf{c}} & = & [\![\overline{T}]\!]_{\mathsf{c}}\otimes[\![S]\!]_{\mathsf{c}}\\
[\![\&\{l_i:S_i\}_{i\in I}]\!]_{\mathsf{su}} & = & ?^{\mathsf{o}}_{\kappa}[\langle l_i:[\![S_i]\!]_{\mathsf{su}}\rangle_{i\in I}] & [\![\&\{l_i:S_i\}_{i\in I}]\!]_{\mathsf{c}} & = & \&\{l_i:[\![S_i]\!]_{\mathsf{c}}\}_{i\in I}\\
[\![\oplus\{l_i:S_i\}_{i\in I}]\!]_{\mathsf{su}} & = & !^{\mathsf{o}}_{\kappa}[\langle l_i:[\![\overline{S_i}]\!]_{\mathsf{su}}\rangle_{i\in I}] & [\![\oplus\{l_i:S_i\}_{i\in I}]\!]_{\mathsf{c}} & = & \oplus\{l_i:[\![S_i]\!]_{\mathsf{c}}\}_{i\in I}
\end{array}
$$

Figure 7: Encodings of session types into usage types (Left) and C-types (Right).

Encoding of Types. We formally relate session types and logic propositions to usage types by means of the encodings given in Figure 7. The former one, denoted as denoted $[\![\cdot]\!]_{\mathsf{su}}$, is taken from [8].

Definition 3.9. *Let Γ be a session typing context. The encoding $[\![\cdot]\!]_f$ into usage typing context and $[\![\cdot]\!]_{\mathsf{c}}$ into C-typing context is inductively defined as follows:*

$$[\![\emptyset]\!]_f = [\![\emptyset]\!]_{\mathsf{c}} \triangleq \emptyset \qquad [\![\Gamma,x:T]\!]_f \triangleq [\![\Gamma]\!]_f, f_x:[\![T]\!]_{\mathsf{su}} \qquad [\![\Gamma,x:T]\!]_{\mathsf{c}} \triangleq [\![\Gamma]\!]_{\mathsf{c}},x:[\![T]\!]_{\mathsf{c}}$$

Lemma 3.10 (Duality and encoding of session types)**.** *Let T,S be finite session types. Then: (i) $\overline{T}=S$ if and only if $\overline{[\![T]\!]_{\mathsf{c}}}=[\![S]\!]_{\mathsf{c}}$; (ii) $\overline{T}=S$ if and only if $\overline{[\![T]\!]_{\mathsf{su}}}=[\![S]\!]_{\mathsf{su}}$.*

On Deadlock Freedom by Encoding. The next results relate deadlock freedom, typing and encoding.

Proposition 3.11. *Let P be a deadlock-free session process, then $[\![P]\!]_f$ is a deadlock-free π-process.*

Proof. Follows by the encoding of terms given in Figure 6, Definition 2.5 and Definition 3.3. □

Next we recall an important result relating deadlock freedom and typing, by following [4].

Corollary 3.12. *Let $\vdash_{\mathtt{ST}} P$ be a session process. If $\vdash^{n}_{\mathtt{KB}} [\![P]\!]_f$ is deadlock-free then P is deadlock-free.*

4 A Hierarchy of Deadlock-Free Session Typed Processes

Preliminaries. To formally define the classes $\mathscr{L}$ and $\mathscr{K}$, we require some auxiliary definitions. The following translation addresses minor syntactic differences between session typed processes (cf. § 2) and the processes typable in the linear logic interpretation of session types (cf. § 3.1). Such differences concern output actions and the restriction operator:

Definition 4.1. *Let P be a session process. The translation $\{\!\{\cdot\}\!\}$ is defined as*

$$\{\!\{\overline{x}\langle y\rangle.P\}\!\} = \overline{x}(z).([z\leftrightarrow y]\mid\{\!\{P\}\!\}) \qquad \{\!\{(\nu xy)P\}\!\} = (\nu w)\{\!\{P\}\!\}[w/x][w/y] \quad w\notin \mathsf{fn}(P)$$

and as an homomorphism for the other process constructs.

Let $[\![\cdot]\!]_{\mathsf{c}}$ denote the encoding of session types into linear logic propositions in Figure 7 (right). Recall that $[\![\cdot]\!]_f$ stands for the encoding of processes and $[\![\cdot]\!]_{\mathsf{su}}$ for the encoding of types, both defined in [8], and given here in Figure 6 and Figure 7 (left), respectively. We may then formally define the languages under comparison as follows:

Definition 4.2 (Typed Languages)**.** *The languages $\mathscr{L}$ and $\mathscr{K}_n$ ($n \geq 0$) are defined as follows:*

$$\begin{array}{rcl} \mathscr{L} & = & \{P \mid \exists \Gamma.\ (\Gamma \vdash_{\mathtt{ST}} P \wedge \{\!\{P\}\!\} \vdash_{\mathtt{CH}} [\![\Gamma]\!]_{\mathsf{c}})\} \\ \mathscr{K}_n & = & \{P \mid \exists \Gamma, f.\ (\Gamma \vdash_{\mathtt{ST}} P \wedge [\![\Gamma]\!]_f \vdash^n_{\mathtt{KB}} [\![P]\!]_f)\} \end{array}$$

Main Results. Our first observation is that there are processes in $\mathscr{K}_2$ but not in $\mathscr{K}_1$:

Lemma 4.3. $\mathscr{K}_1 \subset \mathscr{K}_2$.

Proof. $\mathscr{K}_2$ contains (deadlock-free) session processes not captured in $\mathscr{K}_1$. A representative example is:

$$P_2 = (\nu a_1 b_1)(\nu a_2 b_2)(a_1(x).\ \overline{a_2}\langle x\rangle \mid \overline{b_1}\langle \mathbf{n}\rangle.\ b_2(z))$$

This process is not in $\mathscr{K}_1$ because it involves the composition of two parallel processes which share two sessions. As such, it is typable in $\vdash^n_{\mathtt{KB}}$ (with $n \geq 2$) but not in $\vdash^1_{\mathtt{KB}}$. □

The previous result generalizes easily, so as to define a hierarchy of deadlock-free, session processes:

Theorem 4.4. *For all $n \geq 1$, we have that $\mathscr{K}_n \subset \mathscr{K}_{n+1}$.*

Proof. Immediate by considering one of the following processes, which generalize process P_2 in Lemma 4.3:

$$\begin{array}{rcl} P_{n+1} & = & (\nu a_1 b_1)(\nu a_2 b_2)\cdots(\nu a_{n+1} b_{n+1})(a_1(x).\ \overline{a_2}\langle x\rangle.\cdots.\ \overline{a_{n+1}}\langle y\rangle \mid \overline{b_1}\langle \mathbf{n}\rangle.\ b_2(z).\ \cdots\ b_{n+1}(z)) \\ Q_{n+1} & = & (\nu a_1 b_1)(\nu a_2 b_2)\cdots(\nu a_{n+1} b_{n+1})(a_1(x).\ \overline{a_2}\langle x\rangle.\cdots.\ a_{n+1}(y) \mid \overline{b_1}\langle \mathbf{n}\rangle.\ b_2(z).\ \cdots\ \overline{b_{n+1}}\langle \mathbf{n}\rangle) \end{array}$$

To distinguish $\mathscr{K}_{n+1}$ from $\mathscr{K}_n$, we consider P_{n+1} if $n+1$ is even and Q_{n+1} otherwise. □

One main result of this paper is that $\mathscr{L}$ and $\mathscr{K}_1$ coincide. Before stating this result, we make the following observations. The typing rules for processes in $\mathscr{L}$ do not directly allow free output. However, free output is representable (and typable) by linear logic types by means of the transformation in Definition 4.1. Thus, considered processes are not syntactically equal. In $\mathscr{L}$ there is cooperating composition (enabled by rule (T-cut) in Figure 3); independent composition can only be enabled by rule (T-mix). Arbitrary restriction is not allowed; only restriction of parallel processes.

The following property is key in our developments: it connects our encodings of (dual) session types into usage types with reliability (Definition 3.5), a central notion to the type system for deadlock freedom in Figure 5. Recall that, unlike usage types, there is no parallel composition operator at the level of session types.

Proposition 4.5. *Let T be a session type. Then* $\mathsf{rel}([\![T]\!]_{\mathsf{su}} \mid [\![\overline{T}]\!]_{\mathsf{su}})$ *holds.*

Proof (Sketch). By induction on the structure of session type T and the definitions of $[\![\cdot]\!]_{\mathsf{su}}$ and predicate $\mathsf{rel}(\cdot)$, using Lemma 3.10 (encodings of types preserve session type duality). See [9] for details. □

We then have the following main result, whose proof is detailed in [9]:

Theorem 4.6. $\mathscr{L} = \mathscr{K}_1$.

Therefore, we have the following corollary, which attests that the class of deadlock-free session processes naturally induced by linear logic interpretations of session types is strictly included in the class induced by the indirect approach of Dardha et al. [8] (cf. § 3.2).

Corollary 4.7. $\mathscr{L} \subset \mathscr{K}_n$, $n > 1$.

The fact that (deadlock-free) processes such as P_2 (cf. Lemma 4.3) are not in $\mathscr{L}$ is informally discussed in [3, §6]. However, [3] gives no formal comparisons with other classes of deadlock-free processes.

5 Rewriting $\mathscr{K}_n$ into $\mathscr{L}$

The hierarchy of deadlock-free session processes established by Theorem 4.4 is *subtle* in the following sense: if $P \in \mathscr{K}_{k+1}$ but $P \notin \mathscr{K}_k$ (with $k \geq 1$) then we know that there is a subprocess of P that needs to be "adjusted" in order to "fit in" $\mathscr{K}_k$. More precisely, we know that such a subprocess of P must become more independent in order to be typable under the lesser degree of sharing k.

Here we propose a *rewriting procedure* that converts processes in $\mathscr{K}_n$ into processes in $\mathscr{K}_1$ (that is, $\mathscr{L}$, by Theorem 4.6). The rewriting procedure follows a simple idea: given a parallel process as input, return as output a process in which one of the components is kept unchanged, but the other is replaced by parallel representatives of the sessions implemented in it. Such parallel representatives are formally defined as characteristic processes and catalyzers, introduced next. The rewriting procedure is type preserving and satisfies operational correspondence (cf. Theorems 5.8 and 5.10).

5.1 Preliminaries: Characteristic Processes and Catalyzers

Before presenting our rewriting procedure, let us first introduce some preliminary results.

Definition 5.1 (Characteristic Processes of a Session Type)**.** *Let T be a session type (cf. § 2). Given a name x, the set of* characteristic processes *of T, denoted $\{\!|T|\!\}^x$, is inductively defined as follows:*

$$\begin{array}{rcl}
\{\!|\mathbf{end}|\!\}^x & = & \{P \mid P \vdash_{\mathtt{CH}} x\colon \bullet\} \\
\{\!|?T.S|\!\}^x & = & \{x(y).P \mid P \vdash_{\mathtt{CH}} y\colon [\![T]\!]_{\mathtt{c}}, x\colon [\![S]\!]_{\mathtt{c}}\} \\
\{\!|!T.S|\!\}^x & = & \{\overline{x}(y).(P \mid Q) \mid P \in \{\!|\overline{T}|\!\}^y \wedge Q \in \{\!|S|\!\}^x\} \\
\{\!|\&\{l_i : S_i\}_{i\in I}|\!\}^x & = & \{x \triangleright \{l_i : P_i\}_{i\in I} \mid \forall i \in I.\ P_i \in \{\!|S_i|\!\}^x\} \\
\{\!|\oplus\{l_i : S_i\}_{i\in I}|\!\}^x & = & \bigcup_{i\in I} \{x \triangleleft l_i.P_i \mid P_i \in \{\!|S_i|\!\}^x\}
\end{array}$$

Definition 5.2 (Catalyzer)**.** *Given a session typing context Γ, we define its associated* catalyzer *as a process context $\mathscr{C}_\Gamma[\cdot]$, as follows:*

$$\mathscr{C}_\emptyset[\cdot] = [\cdot] \qquad \mathscr{C}_{\Gamma,x:T}[\cdot] = (\nu x)(\mathscr{C}_\Gamma[\cdot] \mid P) \quad \textit{with } P \in \{\!|\overline{T}|\!\}^x$$

We record the fact that characteristic processes are well-typed in the system of § 3.1:

Lemma 5.3. *Let T be a session type. For all $P \in \{\!|T|\!\}^x$, we have: $P \vdash_{\mathtt{CH}} x : [\![T]\!]_{\mathtt{c}}$*

We use $\{\!|T|\!\}^x \vdash_{\mathtt{CH}} x : [\![T]\!]_{\mathtt{c}}$ to denote the set of processes $P \in \{\!|T|\!\}^x$ such that $P \vdash_{\mathtt{CH}} x : [\![T]\!]_{\mathtt{c}}$.

Lemma 5.4 (Catalyzers Preserve Typability)**.** *Let $\Gamma \vdash_{\mathtt{ST}} P$ and $\Gamma' \subseteq \Gamma$. Then $\mathscr{C}_{\Gamma'}[P] \vdash_{\mathtt{CH}} [\![\Gamma]\!]_{\mathtt{c}} \setminus [\![\Gamma']\!]_{\mathtt{c}}$.*

Corollary 5.5. *Let $\Gamma \vdash_{\mathtt{ST}} P$. Then $\mathscr{C}_\Gamma[P] \vdash_{\mathtt{CH}} \emptyset$.*

5.2 Rewriting $\mathscr{K}_n$ in $\mathscr{L}$

We start this section with some notations. First, in order to represent pseudo-non deterministic binary choices between two equally typed processes, we introduce the following:

Notation 5.6. *Let P_1, P_2 be two processes such that $k \notin \mathsf{fn}(P_1,P_2)$. We write $P_1 \parallel_k P_2$ to stand for the process $(\nu k)(k \triangleleft \mathtt{inx}.\mathbf{0} \mid k \triangleright \{\mathtt{inl}:P_1, \mathtt{inr}:P_2\})$, where label $\mathtt{inx}$ stands for either $\mathtt{inl}$ or $\mathtt{inr}$.*

Clearly, since session execution is purely deterministic, notation $P_1 \parallel_k P_2$ denotes that either P_1 or P_2 will be executed (and that the actual deterministic choice is not relevant). It is worth adding that Caires has already developed the technical machinery required to include non deterministic behavior into the linear logic interpretation of session types; see [1]. Casting our rewriting procedure into the typed framework of [1], so as to consider actual non deterministic choices, is interesting future work.

We find it convenient to annotate bound names in processes with session types, and write $(\nu xy:T)P$ and $x(y:T).P$, for some session type T. When the reduction relation involves a left or right choice in a binary labelled choice, as in reductions due to pseudo-non deterministic choices (Notation 5.6), we sometimes annotate the reduction as $\rightarrow^{\mathtt{inl}}$ or $\rightarrow^{\mathtt{inr}}$. We let $\mathtt{C}$ denote a *process context*, i.e., a process with a hole. And finally, for a typing context Γ, we shall write $\{\!|\Gamma|\!\}$ to denote the process $\prod_{(w_i:T_i)\in\Gamma} \{\!|T_i|\!\}^{w_i}$. We are now ready to give the rewriting procedure from $\mathscr{K}_n$ to $\mathscr{L}$.

Definition 5.7 (Rewriting $\mathscr{K}_n$ into $\mathscr{L}$)**.** *Let $P \in \mathscr{K}_n$ such that $\Gamma \vdash_{\mathtt{ST}} P$, for some Γ. The encoding $(\!|\Gamma \vdash_{\mathtt{ST}} P|\!)$ is a process of $\mathscr{L}$ inductively defined as follows:*

$$
\begin{array}{rll}
(\!|x:\mathbf{end} \vdash_{\mathtt{ST}} \mathbf{0}|\!) \triangleq \mathbf{0} & & \\
(\!|\Gamma \vdash_{\mathtt{ST}} \overline{x}\langle v\rangle.P'|\!) \triangleq \overline{x}(z).\big([v\leftrightarrow z] \mid (\!|\Gamma', x:S \vdash_{\mathtt{ST}} P'|\!)\big) & & \Gamma = \Gamma', x:!T.S, v:T \\
(\!|\Gamma \vdash_{\mathtt{ST}} x(y:T).P'|\!) \triangleq x(y).(\!|\Gamma', x:S, y:T \vdash_{\mathtt{ST}} P'|\!) & & \Gamma = \Gamma', x:?T.S \\
(\!|\Gamma \vdash_{\mathtt{ST}} x \triangleleft l_j.P'|\!) \triangleq x \triangleleft l_j.(\!|\Gamma', x:S_j \vdash_{\mathtt{ST}} P'|\!) & & \Gamma = \Gamma', x:\oplus\{l_i:S_i\}_{i\in I} \\
(\!|\Gamma \vdash_{\mathtt{ST}} x \triangleright \{l_i:P_i\}_{i\in I}|\!) \triangleq x \triangleright \{l_i:(\!|\Gamma', x:S_i \vdash_{\mathtt{ST}} P_i|\!)\}_{i\in I} & & \Gamma = \Gamma', x:\&\{l_i:S_i\}_{i\in I} \\
(\!|\Gamma \vdash_{\mathtt{ST}} (\nu \widetilde{xy}:\widetilde{S})(P \mid Q)|\!) \triangleq \{\!|\Gamma_2|\!\} \mid \mathscr{C}_{\widetilde{z}:\widetilde{S}}\big[(\!|\Gamma_1, \widetilde{x}{:}\widetilde{S} \vdash_{\mathtt{ST}} P|\!)\big[\widetilde{z}/\widetilde{x}\big]\big] & & \Gamma = \Gamma_1 \circ \Gamma_2 \ \wedge \Gamma_1, \widetilde{x}:\widetilde{S} \vdash_{\mathtt{ST}} P \\
\qquad \parallel_k \{\!|\Gamma_1|\!\} \mid \mathscr{C}_{\widetilde{z}:\widetilde{V}}\big[(\!|\Gamma_2, \widetilde{y}{:}\widetilde{V} \vdash_{\mathtt{ST}} Q|\!)\big[\widetilde{z}/\widetilde{y}\big]\big] & & \Gamma_2, \widetilde{y}:\widetilde{V} \vdash_{\mathtt{ST}} Q \wedge V_i = \overline{S_i}
\end{array}
$$

We illustrate the procedure in [9]. Notice that the rewriting procedure given in Definition 5.7 satisfies the compositionality criteria given in [11]. In particular, it is easy to see that the rewriting of a composition of terms is defined in terms of the rewriting of the constituent subterms. Indeed, e.g., $(\!|\Gamma_1 \circ \Gamma_2 \vdash_{\mathtt{ST}} (\nu xy:S)(P \mid Q)|\!)$ depends on a context including both $(\!|\Gamma_1, x:S \vdash_{\mathtt{ST}} P|\!)$ and $(\!|\Gamma_2, y:\overline{S} \vdash_{\mathtt{ST}} Q|\!)$.

We present two important results about our rewriting procedure. First, we show it is type preserving:

Theorem 5.8 (Rewriting is Type Preserving)**.** *Let $(\Gamma \vdash_{\mathtt{ST}} P) \in \mathscr{K}_n$. Then, $(\!|\Gamma \vdash_{\mathtt{ST}} P|\!) \vdash_{\mathtt{CH}} [\![\Gamma]\!]_{\mathtt{c}}$.*

Notice that the inverse of the previous theorem is trivial by following the definition of typed encoding. Theorem 5.8 is meaningful, for it says that the type interface of a process (i.e., the set of sessions implemented in it) is not modified by the rewriting procedure. That is, the procedure modifies the process structure by closely following the causality relations described by (session) types. Notice that causality relations present in processes, but not described at the level of types, may be removed.

The rewriting procedure also satisfies an operational correspondence result. Let us write $\Gamma \vdash_{\mathtt{ST}} P_1, P_2$ whenever both $\Gamma \vdash_{\mathtt{ST}} P_1$ and $\Gamma \vdash_{\mathtt{ST}} P_2$ hold. We have the following auxiliary definition:

Definition 5.9. *Let P, P' be such that $\Gamma \vdash_{\mathtt{ST}} P, P'$. Then, we write $P \div P'$ if and only if $P = \mathtt{C}[Q]$ and $P' = \mathtt{C}[Q']$, for some context $\mathtt{C}$, and there is Γ' such that $\Gamma' \vdash_{\mathtt{ST}} Q, Q'$.*

Theorem 5.10 (Operational Correspondence)**.** *Let $P \in \mathscr{K}_n$ such that $\Gamma \vdash_{\mathtt{ST}} P$ for some Γ. Then we have:*

I) If $P \to P'$ then there exist Q, Q' s.t. (i) $(\!|\Gamma \vdash_{\mathtt{ST}} P|\!) \to^{\mathtt{inx}} \to^* \equiv Q$; (ii) $Q \div Q'$; (iii) $(\!|\Gamma \vdash_{\mathtt{ST}} P'|\!) \to^{\mathtt{inx}} Q'$.

II) If $(\!|\Gamma \vdash_{\mathtt{ST}} P|\!) \to^{\mathtt{inx}} \to^* \equiv Q$ then there exists P' s.t. $P \to P'$ and $Q \div (\!|\Gamma \vdash_{\mathtt{ST}} P'|\!)$.

6 Concluding Remarks

We have presented a formal comparison of fundamentally distinct type systems for deadlock-free, session typed processes. To the best of our knowledge, ours is the first work to establish precise relationships of this kind. Indeed, prior comparisons between type systems for deadlock freedom are informal, given in terms of representative examples typable in one type system but not in some other.

An immediate difficulty in giving a unified account of different typed frameworks for deadlock freedom is the variety of process languages, type structures, and typing rules that define each framework. Indeed, our comparisons involve: the framework of session processes put forward by Vasconcelos [19]; the interpretation of linear logic propositions as session types by Caires [1]; the π-calculus with usage types defined by Kobayashi in [13]. Finding some common ground for comparing these three frameworks is not trivial—several translations/transformations were required in our developments to account for numerous syntactic differences. We made an effort to follow the exact definitions in each framework. Overall, we believe that we managed to concentrate on essential semantic features of two salient classes of deadlock-free session processes, noted $\mathscr{L}$ and $\mathscr{K}$.

Our main contribution is identifying the *degree of sharing* as a subtle, important issue that underlies both session typing and deadlock freedom. We propose a simple characterization of the degree of sharing: in essence, it arises via an explicit premise for the typing rule for parallel composition in the type system in [13]. The degree of sharing is shown to effectively induce a strict hierarchy of deadlock-free session processes in $\mathscr{K}$, as resulting from the approach of [8]. We showed that the most elementary (and non trivial) member of this hierarchy precisely corresponds to $\mathscr{L}$–arguably the most canonical class of session typed processes known to date. Furthermore, by exhibiting an intuitive rewriting procedure of processes in $\mathscr{K}$ into processes in $\mathscr{L}$, we demonstrated that the degree of sharing is a subtle criteria for distinguishing deadlock-free processes. As such, even if our technical developments are technically simple, in our view they substantially clarify our understanding of type systems for liveness properties (such as deadlock freedom) in the context of π-calculus processes.

As future work, we would like to obtain *semantic characterizations* of the degree of sharing, in the form of, e.g., preorders on typed processes that distinguish when one process "is more parallel" than another. We plan also to extend our formal relationships to cover typing disciplines with *infinite behavior*. We notice that the approach of [8] extends to recursive behavior [7] and that infinite (yet non divergent) behavior has been incorporated into logic-based session types [18]. Finally, we plan to explore whether the rewriting procedure given in § 5 could be adapted into a *deadlock resolution* procedure.

Acknowledgements. We are grateful to Luís Caires, Simon J. Gay, and the anonymous reviewers for their valuable comments and suggestions. This work was partially supported by the EU COST Action IC1201 (Behavioural Types for Reliable Large-Scale Software Systems). Dardha is supported by the UK EPSRC project EP/K034413/1 (From Data Types to Session Types: A Basis for Concurrency and Distribution). Pérez is also affiliated to NOVA Laboratory for Computer Science and Informatics, Universidade Nova de Lisboa, Portugal.

References

[1] Luís Caires (2014): *Types and Logic, Concurrency and Non-Determinism*. In *Essays for the Luca Cardelli Fest* - Microsoft Research Technical Report MSR-TR-2014-104. Available at `http://research.microsoft.com/apps/pubs/default.aspx?id=226237`.

[2] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In: *Proc. of CONCUR 2010, LNCS* 6269, Springer, pp. 222–236, doi:10.1007/978-3-642-15375-4_16.

[3] Luís Caires, Frank Pfenning & Bernardo Toninho (2014): *Linear Logic Propositions as Session Types*. *MSCS*, doi:10.1017/S0960129514000218.

[4] Marco Carbone, Ornela Dardha & Fabrizio Montesi (2014): *Progress as Compositional Lock-Freedom*. In: *COORDINATION, LNCS* 8459, Springer, pp. 49–64, doi:10.1007/978-3-662-43376-8_4.

[5] Marco Carbone & Søren Debois (2010): *A Graphical Approach to Progress for Structured Communication in Web Services*. In: *Proc. of ICE 2010, Amsterdam, The Netherlands, 10th of June 2010., EPTCS* 38, pp. 13–27, doi:10.4204/EPTCS.38.4.

[6] Gerardo Costa & Colin Stirling (1987): *Weak and Strong Fairness in CCS*. *Inf. Comput.* 73(3), pp. 207–244, doi:10.1016/0890-5401(87)90013-7.

[7] Ornela Dardha (2014): *Recursive Session Types Revisited*. In: *Proceedings Third Workshop on Behavioural Types, BEAT 2014, Rome, Italy, 1st September 2014., EPTCS* 162, pp. 27–34, doi:10.4204/EPTCS.162.4.

[8] Ornela Dardha, Elena Giachino & Davide Sangiorgi (2012): *Session types revisited*. In: *Proc. of PPDP'12*, ACM, pp. 139–150, doi:10.1145/2370776.2370794.

[9] Ornela Dardha & Jorge A. Pérez (2015): *Full version of this paper*. Technical Report. Available at `http://www.jorgeaperez.net`.

[10] Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro & Nobuko Yoshida (2008): *On Progress for Structured Communications*. In: *Trustworthy Global Computing, LNCS* 4912, Springer, pp. 257–275, doi:10.1007/978-3-540-78663-4_18.

[11] Daniele Gorla (2010): *Towards a unified approach to encodability and separation results for process calculi*. *Inf. Comput.* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[12] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In: *Proc. of ESOP'98, LNCS* 1381, Springer, pp. 122–138, doi:10.1007/BFb0053567.

[13] Naoki Kobayashi (2002): *A Type System for Lock-Free Processes*. *Inf. Comput.* 177(2), pp. 122–159, doi:10.1006/inco.2002.3171.

[14] Naoki Kobayashi (2003): *Type Systems for Concurrent Programs*. In: *Formal Methods at the Crossroads, LNCS* 2757, Springer, pp. 439–453, doi:10.1007/978-3-540-40007-3_26.

[15] Naoki Kobayashi (2006): *A New Type System for Deadlock-Free Processes*. In: *Proc. of CONCUR 2006, LNCS* 4137, Springer, pp. 233–247, doi:10.1007/11817949_16.

[16] Luca Padovani (2013): *From Lock Freedom to Progress Using Session Types*. In: *Proceedings of PLACES 2013, Rome, Italy, 23rd March 2013., EPTCS* 137, pp. 3–19, doi:10.4204/EPTCS.137.2.

[17] Benjamin C. Pierce (2002): *Types and programming languages*. MIT Press, MA, USA.

[18] Bernardo Toninho, Luís Caires & Frank Pfenning (2014): *Corecursion and Non-divergence in Session-Typed Processes*. In: *Proc. of TGC 2014, LNCS* 8902, Springer, pp. 159–175, doi:10.1007/978-3-662-45917-1_11.

[19] Vasco T. Vasconcelos (2012): *Fundamentals of session types*. *Inf. Comput.* 217, pp. 52–70, doi:10.1016/j.ic.2012.05.002.

[20] Hugo Torres Vieira & Vasco Thudichum Vasconcelos (2013): *Typing Progress in Communication-Centred Systems*. In: *COORDINATION, LNCS* 7890, Springer, pp. 236–250, doi:10.1007/978-3-642-38493-6_17.

[21] Philip Wadler (2012): *Propositions as sessions*. In: *Proc. of ICFP'12*, pp. 273–286, doi:10.1145/2364527.2364568.

On Compensation Primitives as Adaptable Processes

Jovana Dedeić
University of Novi Sad, Serbia

Jovanka Pantović
University of Novi Sad, Serbia

Jorge A. Pérez
University of Groningen, The Netherlands

We compare mechanisms for *compensation handling* and *dynamic update* in calculi for concurrency. These mechanisms are increasingly relevant in the specification of reliable communicating systems. Compensations and updates are intuitively similar: both specify how the behavior of a concurrent system changes at runtime in response to an exceptional event. However, calculi with compensations and updates are technically quite different. We investigate the *relative expressiveness* of these calculi: we develop encodings of core process languages with compensations into a calculus of *adaptable processes* developed in prior work. Our encodings shed light on the (intricate) semantics of compensation handling and its key constructs. They also enable the transference of existing verification and reasoning techniques for adaptable processes to core languages with compensation handling.

1 Introduction

Many software applications are based on *long-running transactions* (LRTs). Frequently found in service-oriented systems [8], LRTs are computing activities which extend in time and may involve distributed, loosely coupled resources. These features sharply distinguish LRTs from usual (database) transactions. One particularly delicate aspect of LRTs management is handling (partial) failures: mechanisms for detecting failures and bringing the LRT back to a consistent state need to be explicitly programmed. As designing and certifying the correctness of such mechanisms is error prone, the last decade has seen the emergence of specialized constructs, such as *exceptions* and *compensations*, which offer direct programming support. Our focus is in the latter: as their name suggests, compensation mechanisms are meant to compensate the fact that an LRT has failed or has been aborted. Upon reception of an abortion or failure signal, compensation mechanisms are expected to install and activate alternative behaviors for recovering system consistency. Such a compensation behavior may be different from the LRT's initial behavior.

A variety of calculi for concurrency with constructs for compensation handling has been proposed (see, e.g., [1, 5, 8, 14]). Building upon the tradition and approach of mobile process calculi such as the π-calculus [16], they capture different forms of error recovery and offer reasoning techniques (e.g., behavioral equivalences) on communicating processes with compensation constructs. The relative expressive power of such proposals has also been studied [4, 5, 12, 13]. On a related but different vein, a calculus of *adaptable processes* has been put forward as a process calculus approach to specify the dynamic evolution of interacting systems [2]. It is intended as a way of overcoming the limitations that process calculi have for describing patterns of dynamic evolution. In this calculus, process behaviors may be enclosed by nested, transparent *locations*; actions of dynamic update are targeted to particular locations. This model allows us to represent a wide range of evolvability patterns for concurrent processes. The theory of adaptable processes includes expressiveness, decidability, and verification results [2, 3], as well as the integration with structured communications governed by session types [9, 10].

Adaptable processes specify forms of dynamic reconfiguration which are triggered by exceptional events, not necessarily catastrophic. For instance, an external request for upgrading a working component is an exceptional event which is hard to predict and entails a modification of the system's behavior. Still, it is certainly not an error or a failure. Thus, adaptation intuitively appears to us as a general phenomenon

S. Crafa and D. Gebler (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2015)
EPTCS 190, 2015, pp. 16–30, doi:10.4204/EPTCS.190.2

© J. Dedeić, J. Pantović & J. A. Pérez
This work is licensed under the
Creative Commons Attribution License.

which includes the (negative) exceptional events dealt by compensations. That is, it should be possible to represent failures and compensation activities as particular instances of the behaviors expressible in [2].

In this paper, we make this intuitive observation precise by encoding calculi with compensations into adaptable processes. Our motivation is twofold. First, given the diversity of linguistic constructs for compensations, understanding how they can be implemented as adaptable processes could shed new light in their formal underpinnings. Since adaptable processes have a simple semantics (based on higher-order process communication [17]), the envisaged encodings could suggest alternative semantics for existing formalisms. Second, given that adaptable processes have been developed in several directions, encodings of calculi with compensations into adaptable processes could enable the transference of, e.g., decidability results or type systems, from adaptable processes to calculi with compensations.

As source languages in our study, we systematically consider the different classes of calculi with compensations developed in [12], a work that offers a unified presentation for many calculi proposed in the literature. In particular, we consider processes with *static* and *dynamic* compensations, each of them with *preserving*, *discarding*, and *aborting* semantics. (All these semantics are illustrated next.) As such, we offer six different encodings into adaptable processes, each one equipped with appropriate operational correspondence results. The encodings are rather involved; in particular, representing preserving, discarding, and aborting semantics by means of the transparent locations in [2] proved to be quite challenging. In our view, the intricate character of our representations into adaptable processes is directly related to the intricate semantics of each of the forms of calculi with compensations.

This paper is structured as follows. § 2 illustrates primitives for adaptable processes and compensation handling; § 3 formally presents the corresponding calculi. In § 4 we define and prove correct encodings of processes with static compensations into adaptable processes. We consider aborting, preserving, and discarding semantics. § 5 describes encodings of processes with dynamic compensations. § 6 collects some concluding remarks. Due to space restrictions, omitted proofs can be found online [7].

2 Adaptable and Compensable Processes, By Example

We give an intuitive account of the calculus of *adaptable processes* (introduced by Bravetti et al. [2]) and of the core calculus with primitives for *compensation handling* (as presented by Lanese et al. [12, 13]).

Adaptable Processes. The calculus of *adaptable processes* was introduced in [2] as a variant of Milner's CCS [15] (without restriction and relabeling), extended with the following two constructs, aimed at representing the dynamic reconfiguration (or update) of active communicating processes:

1. A *located process*, denoted $l[P]$, represents a process P which resides in a location called l. Locations are *transparent*: the behavior of $l[P]$ is the same as the behavior of P. Locations can also be arbitrarily *nested*, which allows to organize process descriptions into meaningful hierarchical structures.
2. An *update prefix* $l\{(X).Q\}$—where X is a process variable that occurs zero or more times in Q—denotes an adaptation mechanism for processes located at location l.

This way, in the calculus of adaptable process the possibility of updating a (located) process behavior is given the same status as communication prefixes. Intuitively, an update prefix for location l is able to interact with a located process at l, updating its current behavior. This is captured by the reduction rule

$$C_1\big[l[P]\big] \mid C_2\big[l\{(X).Q\}.R\big] \longrightarrow C_1\big[Q\{^P/_X\}\big] \mid C_2\big[R\big]$$

where C_1 and C_2 denote *contexts* which may describe, e.g., nested locations and parallel components. Therefore, the adaptation mechanism (embodied by $l\{(X).Q\}$) moves to the place where $l[P]$ resides (C_1

above) and exercises a dynamic update there, as represented by substitution $Q\{^{P}/_{X}\}$. As such, adaptation is a form of *higher-order process communication* [17]. Observe that Q may not contain X, so the current behavior at l (i.e., P) may get erased as a result of the update. Notice also that this form of adaptation is *subjective*: located processes are influenced by (unknown) update prefixes in their environment.

Compensable Processes. Our core process language with compensations is based on the calculus in [13] (a variant of the language in [12]). The languages in [12, 13] are appealing because they uniformly capture several different proposals for calculi with compensation handling. These calculi were introduced as extensions of the π-calculus [16] with primitives for *static* and *dynamic recovery*. However, in order to focus on the essentials of compensation handling primitives, in this presentation we consider a variant of the languages in [12, 13] without name mobility. There are three salient constructs:

1. *Transaction scopes* (or simply *transactions*), denoted $t[P\,,Q]$, where t is a name and P, Q are processes;
2. *Protected blocks*, denoted $\langle Q\rangle$, for some process Q;
3. *Compensation updates*, denoted $\mathtt{inst}\lfloor\lambda X.Q\rfloor.P$, where P, Q are processes and X is a process variable that occurs zero or more times in Q.

While transactions and protected blocks define static recovery mechanisms, compensation updates are used to define dynamic recovery. We now gradually introduce these constructs and their main features.

Basic Intuitions. A transaction $t[P\,,Q]$ consists of a *default activity* P with a *compensation activity* Q. Transactions can be nested, so process P in $t[P\,,Q]$ may contain other transactions. Transactions can be aborted: intuitively, process $t[P\,,Q]$ behaves as P until an *error notification* (abortion signal) arrives along name t. Error notifications are simply output messages which can originate inside or outside the transaction. To illustrate the simplest manifestation of compensations, we have the following transitions:

$$t[P\,,Q] \mid \overline{t}.R \xrightarrow{\tau} Q \mid R \qquad\qquad t[\overline{t}.P_1 \mid P_2\,,Q] \mid R \xrightarrow{\tau} Q \mid R$$

While the transition in the left shows how a transaction t can be aborted by an external signal, the transition in the right illustrate abortion due to an internal signal. In both cases, abortion leads to discarding the default behavior of the transition, and the compensation activity is executed instead (Q in both cases).

Protected Blocks. The transitions above illustrate the different sources of abortion signals that lead to compensation behaviors. One key element in calculi with compensations primitives are *protected blocks*: as their name suggests, these constructs protect a process from abortion signals. Similarly as locations, protected blocks are transparent: Q and $\langle Q\rangle$ have the same behavior, but $\langle Q\rangle$ cannot be affected by abortion signals. Protected blocks are meant to prevent abortions after a compensation:

$$t_2[P_2\,,Q_2] \mid \overline{t_2} \xrightarrow{\tau} \langle Q_2\rangle$$

That is, the compensation behavior Q_2 will be immune to external errors thanks to protected blocks. Consider now process $t_1\big[t_2[P_2\,,Q_2] \mid \overline{t_2}.R_1\,,Q_1\big]$, which includes a transaction named t_2 which is *nested* inside t_1. Although in previous examples the default behavior has been erased following an abortion signal, the semantics of compensations actually may partially preserve such behavior. This is realized by *extraction functions*, denoted $\mathtt{extr}(\cdot)$. For the previous process, we have the following transition:

$$t_1\big[t_2[P_2\,,Q_2] \mid \overline{t_2}.R_1 \mid R_2\,,Q_1\big] \xrightarrow{\tau} t_1\big[\langle Q_2\rangle \mid \mathtt{extr}(P_2) \mid R_1\,,Q_1\big]$$

In case transaction t_2 is aborted, its compensation behavior Q_2 will be preserved. Moreover, part of the behavior of P_2 will be preserved as well: this is expressed by process $\mathtt{extr}(P_2)$, which consists of at least all protected blocks in P_2; it may also contain some other processes, related to transactions (see next).

We consider *discarding*, *preserving*, and *aborting* variants for $\mathtt{extr}(\cdot)$; they define three different semantics for compensations. Noted $\mathtt{extr}_{\mathtt{D}}(\cdot)$, $\mathtt{extr}_{\mathtt{P}}(\cdot)$, and $\mathtt{extr}_{\mathtt{A}}(\cdot)$, respectively, these functions concern mostly protected blocks and transactions. Given a process P, we would have:

- $\mathtt{extr}_{\mathtt{D}}(P)$ keeps only protected blocks in P. Other processes (including transactions) are discarded.
- $\mathtt{extr}_{\mathtt{P}}(P)$ keeps protected blocks and transactions at the top-level in P. Other processes are discarded.
- $\mathtt{extr}_{\mathtt{A}}(P)$ keeps protected blocks and nested transactions in P, including their respective compensation activities. Other processes are discarded.

As an example, consider the process $P = t\big[t_1[P_1, Q_1] \mid t_2[\langle P_2\rangle, Q_2] \mid R \mid \langle P_3\rangle, Q_5\big]$. We then have:

$$\begin{array}{lll}
\text{Discarding semantics:} & \overline{t} \mid P & \xrightarrow{\tau}_{\mathtt{D}} \quad \langle P_3\rangle \mid \langle Q_5\rangle \\
\text{Preserving semantics:} & \overline{t} \mid P & \xrightarrow{\tau}_{\mathtt{P}} \quad \langle P_3\rangle \mid \langle Q_5\rangle \mid t_1[P_1, Q_1] \mid t_2[\langle P_2\rangle, Q_2] \\
\text{Aborting semantics:} & \overline{t} \mid P & \xrightarrow{\tau}_{\mathtt{A}} \quad \langle P_3\rangle \mid \langle Q_5\rangle \mid \langle P_2\rangle \mid \langle Q_1\rangle \mid \langle Q_2\rangle
\end{array}$$

Thus, the three different semantics implement different levels of protection. The discarding semantics only concerns the compensation activity for transaction t and the protected block $\langle P_3\rangle$. The preserving semantics protects also the nested transactions t_1 and t_2; a process such as R, without an enclosing protected block, is discarded. Finally, the aborting semantics preserves all protected blocks and compensation activities in the default activity for t, including those in nested transactions, such as $\langle P_2\rangle$.

Dynamic Compensations. Up to here we have considered transactions with *static compensations*: while the default behavior may change due to transaction abortion, the compensable behavior remains unchanged. Given a transaction $t[P, Q]$, using *compensation updates* one may specify in P an update for the compensation behavior Q. This is achieved by the operator $\mathtt{inst}\lfloor \lambda X.Q\rfloor.P$, where $\lambda X.Q$ is a function which represents the compensation update. As a simple example, consider the following transition:

$$t\big[\mathtt{inst}\lfloor \lambda X.R\rfloor.P_1 \mid P_2, Q\big] \xrightarrow{\tau} t\big[P_1 \mid P_2, R\{^{Q}/_{X}\}\big]$$

This way, $\mathtt{inst}\lfloor \lambda X.R\rfloor.P$ produces a new compensation behavior $R\{^{Q}/_{X}\}$ after an internal transition. As variable X may not occur in R, this step may fully discard the previous compensation activity Q.

3 The Calculi

We introduce adaptable processes (§ 3.1) and compensable processes (§ 3.2). To focus on their essentials, both calculi are defined as extensions of CCS [15] (no name passing involved). In both cases, we assume a countable set of names N, ranged over by $a, b, l, t, \ldots$. As a convention, we use names $l, l', \ldots$ to denote locations (in adaptable processes) and names $t, t', \ldots$ to denote transactions (in compensable processes).

3.1 Adaptable Processes

The syntax of the calculus of *adaptable processes* is defined by *prefixes* $\pi, \pi', \ldots$ and *processes* $P, Q, \ldots$:

$$\pi \;::=\; a \;\mid\; \overline{a} \;\mid\; l\{(X).Q\} \qquad\qquad P \;::=\; l[P] \;\mid\; \mathbf{0} \;\mid\; \pi.P \;\mid\; !P \;\mid\; P \mid Q \;\mid\; (\nu a)P \;\mid\; X$$

We consider input and output prefixes (noted a and $\overline{a}$, respectively) and the *update prefix* $l\{(X).Q\}$, where Q may contain zero or more occurrences of *process variable* X. The syntax of processes includes *located processes* (noted $l[P]$ and intuitively motivated above) as well as usual CCS constructs for inaction, prefix

$$\text{(R-I/O)}\quad E\Big[C\big[\overline{a}.P\big] \mid D\big[a.Q\big]\Big] \to E\Big[C\big[P\big] \mid D\big[Q\big]\Big] \qquad \text{(R-UPD)}\quad E\Big[C\big[l[P]\big] \mid D\big[l\{(X).Q\}.R\big]\Big] \to E\Big[C\big[Q\{^P/_X\}\big] \mid D\big[R\big]\Big]$$

$$\text{(R-PAR)}\ \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \text{(R-RES)}\ \frac{P \to P'}{(\nu a)P \to (\nu a)P'} \qquad \text{(R-STR)}\ \frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$$

Figure 1: Reduction semantics for adaptable processes.

(sequentiality), replication, parallel composition, and restriction. We omit $\mathbf{0}$ whenever possible; we write, e.g., $l\{(X).P\}$ instead of $l\{(X).P\}.\mathbf{0}$. Name a is bound in $(\nu a)P$ and process variable X is bound in $l\{(X).Q\}$; given a process P, its sets of free and bound names/variables—denoted $\mathtt{fn}(P)$, $\mathtt{bn}(P)$, $\mathtt{fv}(P)$, and $\mathtt{bv}(P)$—are as expected. We rely on expected notions of α-conversion (noted $\equiv_\alpha$) and process substitution: $P\{^Q/_X\}$ denotes the process obtained by (capture avoiding) substitution of Q for X in P.

The semantics of adaptable processes is given by a reduction semantics, denoted $\to$, and defined as the smallest relation on processes induced by the rules in Figure 1. $\to^*$ denotes the reflexive and transitive closure of $\to$. Reduction relies on *structural congruence*, denoted $\equiv$, and *contexts*, denoted C, D, E. We define $\equiv$ as the smallest congruence on processes that satisfies the axioms:

$P \mid Q \equiv Q \mid P$	$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$	$P \mid \mathbf{0} \equiv P$
$P \equiv Q$ if $P \equiv_\alpha Q$	$(\nu a)\mathbf{0} \equiv \mathbf{0}$	$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$
$(\nu a)P \mid Q \equiv (\nu a)(P \mid Q)$ if $a \notin \mathtt{fn}(Q)$	$(\nu a)l[P] \equiv l[(\nu a)P]$	$!P \equiv P \mid !P$

The syntax of monadic contexts (processes with a single *hole*, denoted $[\bullet]$) is defined as:

$$C ::= [\bullet] \ \Big|\ C \mid P \ \Big|\ l\big[C\big]$$

We write $C[P]$ to denote the process resulting from filling in all occurrences of $[\bullet]$ in context C with process P. We comment on rules in Figure 1. Rule (R-I/O) formalizes synchronization between process $\overline{a}.P$ and process $a.Q$ (enclosed in contexts C and D, respectively). Rule (R-UPD) formalizes the dynamic update/evolvability of a location l. The result of the synchronization between a located process $l[P]$ and an update prefix $l\{(X).Q\}$ is the process $Q\{^P/_X\}$. This resulting process stays in the same context as process $l[P]$. Rules (R-PAR), (R-RES), and (R-STR) are standard and/or self-explanatory.

3.2 Compensable Processes

The calculus of *compensable processes* extends CCS with constructs for transactions, protected blocks, and compensation updates:

$$\pi ::= a \mid \overline{a} \qquad P,Q ::= \mathbf{0} \mid \pi.P \mid !P \mid (\nu a)P \mid P \mid Q \mid t[P,Q] \mid \langle Q \rangle \mid X \mid \mathtt{inst}\lfloor \lambda X.R \rfloor.P$$

Prefixes π include input and output actions. Processes for inaction ($\mathbf{0}$), sequentiality ($\pi.P$), replication ($!P$), restriction ($(\nu a)P$), and parallel composition ($P \mid Q$) are standard. We omit $\mathbf{0}$ whenever possible. Protected blocks $\langle Q \rangle$, transactions $t[P,Q]$, and compensation updates $\mathtt{inst}\lfloor \lambda X.R \rfloor.P$ have been already motivated. Error notifications are simply output messages; they can be internal (coming from the default activity) or external (coming from outside of the transaction). Name a is bound in $(\nu a)P$ and variable X is bound in $\mathtt{inst}\lfloor \lambda X.R \rfloor$; given a process P, its sets of free and bound names/variables—denoted $\mathtt{fn}(P)$, $\mathtt{bn}(P)$, $\mathtt{fv}(P)$, and $\mathtt{bv}(P)$—are as expected. α-conversion (noted $\equiv_\alpha$) and substitution $P\{^Q/_X\}$

$$\mathsf{extr}_{\mathtt{D}}(t[P,Q]) = \mathbf{0} \qquad \mathsf{extr}_{\mathtt{P}}(t[P,Q]) = t[P,Q] \qquad \mathsf{extr}_{\mathtt{A}}(t[P,Q]) = \mathsf{extr}_{\mathtt{A}}(P) \mid \langle Q \rangle$$
$$\mathsf{extr}(\langle P \rangle) = \langle P \rangle \qquad \mathsf{extr}(P \mid Q) = \mathsf{extr}(P) \mid \mathsf{extr}(Q) \qquad \mathsf{extr}((\nu a)P) = (\nu a)\mathsf{extr}(P)$$
$$\mathsf{extr}(!P) = \mathbf{0} \qquad \mathsf{extr}(\mathtt{inst}\lfloor \lambda X.Q \rfloor.P) = \mathbf{0} \qquad \mathsf{extr}(\pi.P) = \mathbf{0}$$

Figure 2: Extraction functions.

are also as expected. We assume that protected blocks and transactions do not appear behind prefixes; this is key to ensure encoding correctness. We shall say that the sub-calculus without compensation updates $\mathtt{inst}\lfloor \lambda X.R \rfloor.P$ is the calculus with *static compensations*; the full calculus will be referred to as the calculus with *dynamic compensations*. The following definitions apply uniformly to both.

Following [12, 13], the semantics of compensable processes is given in terms of a Labeled Transition System (LTS). Ranged over α, α', the set of labels includes a, $\overline{a}$, τ, and $\lambda X.Q$. As in CCS, a denotes an input action, $\overline{a}$ denotes an output action, and τ denotes synchronization (internal action). Label $\lambda X.Q$ is associated to compensation updates. Formally, we have three different LTSs, corresponding to processes under discarding, preserving, and aborting semantics. Therefore, for each $\kappa \in \{\mathtt{D}, \mathtt{P}, \mathtt{A}\}$, we will have an extraction function $\mathsf{extr}_\kappa(\cdot)$ and a transition relation $\xrightarrow{\alpha}_\kappa$. The different extraction functions are defined in Fig. 2; the rules of the LTSs are given in Fig. 3. As a convention, whenever a notion coincides for the three semantics, we shall avoid decorations $\mathtt{D}$, $\mathtt{P}$, and $\mathtt{A}$. This way, e.g., by writing $\mathsf{extr}(\langle P \rangle) = \langle P \rangle$ we mean that the extraction function for protected blocks is the same for all three semantics.

We comment on the rules in Fig. 3. Axioms (L-OUT) and (L-IN) execute output and input prefixes, respectively. Rule (L-REP) deals with replication, while rule (L-PAR) allows one parallel component to progress independently. Rule (L-RES) is the standard rule for restriction: it states that a transition of process P determines a transition of process $(\nu a)P$, where label α provides that the restriction name a does not occur inside α. Rule (L-COMM) defines communication on a. Rule (L-SCOPE-OUT) allows the default activity P of a transaction to progress, provided that the performed action is not a compensation update and that there is no pending compensation update to be executed. The latter is ensured by condition $\mathsf{noComp}(\mathsf{P})$, defined in [7]: the condition is true if and only if process P does not have compensation update which waits for execution. This means that a compensation update has priority over other transitions; that is, if process P in transaction $t[P,Q]$ has a compensation update at top-level then it will be performed before any change of the current state. Rule (L-RECOVER-OUT) allows an external process to abort a transaction via an output action $\overline{t}$. The resulting process contains two parts: the first part is obtained from the default activity P of the transaction via the appropriate extraction function; the second part corresponds to compensation Q which will be executed inside a protected block. Similarly, rule (L-RECOVER-IN) handles abortion when the error notification comes from the default activity P of the transaction. Rule (L-BLOCK) essentially specifies that protected blocks are transparent units. Observe that the actual semantics of protected blocks is defined via the extraction functions $\mathsf{extr}(\cdot)$. The final two rules are peculiar of processes with dynamic compensations: while rule (L-INST) performs a compensation update, rule (L-SCOPE-CLOSE) updates the compensation of a transaction.

We find it convenient to define structural congruence ($\equiv$) and contexts also for compensable processes. We define $\equiv$ as the smallest congruence on processes that includes $\equiv_\alpha$ and satisfies the axioms:

$$P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad P \mid \mathbf{0} \equiv P$$
$$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad (\nu a)P \mid Q \equiv (\nu a)(P \mid Q) \text{ if } a \notin \mathtt{fn}(Q) \qquad (\nu a)\mathbf{0} \equiv \mathbf{0}$$
$$\langle\langle P \rangle\rangle \equiv \langle P \rangle \qquad \langle (\nu a)P \rangle \equiv (\nu a)\langle P \rangle \qquad \langle \mathbf{0} \rangle \equiv \mathbf{0}$$
$$t[(\nu a)P, Q] \equiv (\nu a)t[P,Q] \text{ if } t \neq a,\ a \notin \mathtt{fn}(Q) \qquad (\nu a)\overline{a} \equiv \mathbf{0}$$

$$\text{(L-OUT)}\ \ \overline{a}.P \xrightarrow{\overline{a}} P \qquad \text{(L-IN)}\ \ a.P \xrightarrow{a} P \qquad \text{(L-REP)}\ \ \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \text{(L-PAR)}\ \ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \text{(L-RES)}\ \ \frac{P \xrightarrow{\alpha} P' \quad \alpha \neq a, \overline{a}}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'} \qquad \text{(L-COMM)}\ \ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\text{(L-SCOPE-OUT)}\ \ \frac{P \xrightarrow{\alpha} P' \quad \alpha \neq \lambda X.Q \quad \mathsf{noComp(P)}}{t[P,Q] \xrightarrow{\alpha} t[P',Q]} \qquad \text{(L-RECOVER-OUT)}\ \ \frac{\mathsf{noComp(P)}}{t[P,Q] \xrightarrow{t} \mathsf{extr}(P) \mid \langle Q \rangle} \qquad \text{(L-RECOVER-IN)}\ \ \frac{P \xrightarrow{\overline{t}} P' \quad \mathsf{noComp(P)}}{t[P,Q] \xrightarrow{\tau} \mathsf{extr}(P') \mid \langle Q \rangle}$$

$$\text{(L-BLOCK)}\ \ \frac{P \xrightarrow{\alpha} P'}{\langle P \rangle \xrightarrow{\alpha} \langle P' \rangle} \qquad \text{(L-INST)}\ \ \mathtt{inst}\lfloor \lambda X.Q \rfloor.P \xrightarrow{\lambda X.Q} P \qquad \text{(L-SCOPE-CLOSE)}\ \ \frac{P \xrightarrow{\lambda X.R} P'}{t[P,Q] \xrightarrow{\tau} t[P', R\{^{Q}/_{X}\}]}$$

Figure 3: LTS for compensable processes. Symmetric variants of (L-PAR) and (L-COMM) are omitted.

An n-adic context $C[\bullet_1, \ldots, \bullet_n]$ is obtained from a process by replacing n occurrences of $\mathbf{0}$, that are neither compensations nor in continuation of prefixes, with indexed holes $[\bullet_1], \ldots, [\bullet_n]$. This way, for instance, the syntax of monadic contexts is defined as:

$$C ::= [\bullet] \ \mid\ \langle C \rangle \ \mid\ t[C,P] \ \mid\ P \mid C \ \mid\ C \mid P \ \mid\ (\nu a)C.$$

We write $C[P]$ to denote the process resulting from filling in all occurrences of $[\bullet]$ in context C with process P. The following proposition is central to our operational correspondence statements.

Proposition 3.1. *Let P be a compensable process. If $P \xrightarrow{\tau} P'$ then one of the following holds:*

a) $P \equiv E[C[\overline{a}.P_1] \mid D[a.P_2]]$ *and* $P' \equiv E[C[P_1] \mid D[P_2]]$,

b) $P \equiv E[C[t[P_1,Q]] \mid D[\overline{t}.R]]$ *and* $P' \equiv E[C[\mathsf{extr}(P_1) \mid \langle Q \rangle] \mid D[R]]$,

c) $P \equiv C[t[D[\overline{t}.P_1],Q]]$ *and* $P' \equiv C[\mathsf{extr}(D[P_1]) \mid \langle Q \rangle]$,

d) $P \equiv E[t[C'[\mathtt{inst}\lfloor \lambda X.R \rfloor.P], Q]]$ *and* $P' \equiv E[t[C'[P], R\{^{Q}/_{X}\}]]$,

for some contexts C, C', D, E, processes P_1, P_2, Q, R, and names a, t.

4 Encoding Static Compensation Processes

Here we present encodings of processes with static compensations into adaptable processes. We consider discarding, preserving and aborting semantics. We adopt the following abbreviations for update prefixes:

- $t\{\dagger\}$ for the update prefix $t\{(Y).\mathbf{0}\}$ which "kills" location t, together with the process located at t;
- $t\{P\}$ for the update prefix $t\{(Y).P\}$ (with $Y \notin \mathtt{fv}(P)$) that replaces the current behavior at t with P;
- $t\{\mathsf{id}\}$ for the update prefix $t\{(X).X\}$ which deletes the location name t;
- $t\{(X_1, X_2, \ldots, X_n).R\}$ for the sequential composition of updates $t\{(X_1).t\{(X_2).\cdots.t\{(X_n).R\}\}\}$.

Basic Intuitions. We describe some commonalities in the encodings we are about to present. Unsurprisingly, the main challenge to encodability is in representing transactions $t[P,Q]$ and protected blocks $\langle R \rangle$ as adaptable processes. Our strategy consists in representing P and Q independently, using located

processes. Since locations are transparent units of behavior, this suffices for encoding P. However, the encoding of Q cannot freely execute unless an abortion signal (an output action) is received. Very approximately, our encodings of protected blocks and transactions have the following structure:

$$[\![\langle R\rangle]\!]_{t,\rho} = p_{t,\rho}\left[\,[\![R]\!]_{\varepsilon}\,\right] \quad (1)$$

$$[\![t[P,Q]]\!]_{\rho} = \underbrace{t\left[\,[\![P]\!]_{t,\rho}\,\right]}_{\text{(a)}} \mid \underbrace{l_t.\pi_1.\cdots.\pi_k.p_t\left[\,[\![Q]\!]_{t,\rho}\,\right]}_{\text{(b)}} \mid \underbrace{t.\overline{l_t}.K}_{\text{(c)}} \quad (2)$$

In our encodings we use *paths*, finite sequences of names, denoted $t_1, t_2, \ldots, t_n$. The empty path is denoted ε. Ranged over ρ, paths capture the hierarchical structure of nested transactions. Using paths, for each protected block, we maintain an association with the name of its enclosing transaction. As such, the encoding of a protected block associated to transaction t will be enclosed in a location p_t (see (1) above). There could be more than one occurrence of such locations, as the transaction's body may contain several protected blocks. The encoding of transactions, given in (2), consists of three parallel components:

- Component (a) is a location which contains the encoding of the default activity of the transaction; we retain the name of the transaction in the source process.
- Component (b) represents the compensation activity of the transaction. It is given as a located process at p_t, and is protected by a number of prefixes $\pi_1, \cdots, \pi_k$ including an input prefix l_t.
- Component (c) handles abortion signals. After synchronizing with an output on t, it synchronizes with the input on l_t in component (b). This releases a process K which "collects" all protected blocks in the encoding of P (which occur inside locations named p_t) but also the encoding of the compensation activity Q. This collection process may involve synchronizations with $\pi_1, \cdots, \pi_k$ in (b). Once all protected blocks have been collected, location t is destroyed.

This (very approximate) strategy is used in all of our encodings, with variations motivated by discarding, preserving, and aborting semantics. Knowing the number of protected blocks to be collected is crucial in this scheme. To this end, appropriate counting functions on the default activity P are defined.

The following remark defines some basic conditions on "reserved names" used in our encodings:

Remark 4.1. *Let t be a name, then we know that there are names l_t, k_t, p_t and m_t which are associated with the name t. Also, if $t_1 \neq t_2$ then $l_{t_1} \neq l_{t_2}, k_{t_1} \neq k_{t_2}, p_{t_1} \neq p_{t_2}$ and $m_{t_1} \neq m_{t_2}$.*

4.1 Discarding Semantics

Before presenting the encoding, we introduce some auxiliary functions. First, we introduce a function that counts the number of protected blocks in a process.

Definition 4.2 (Number of protected blocks)**.** *Let P be a compensable process. The number of protected blocks in P, denoted by $\mathtt{npb}_{\mathtt{D}}(P)$, is defined as follows:*

$$\mathtt{npb}_{\mathtt{D}}(P) = \begin{cases} 1 & \text{if } P = \langle P_1\rangle \\ \mathtt{npb}_{\mathtt{D}}(P_1) + \mathtt{npb}_{\mathtt{D}}(P_2) & \text{if } P = P_1 \mid P_2 \\ \mathtt{npb}_{\mathtt{D}}(P_1) & \text{if } P = (\nu a)P_1 \\ 0 & \text{otherwise.} \end{cases}$$

We shall define an encoding $\mathtt{D}[\![\cdot]\!]_{\rho}$ of compensable processes into adaptable processes, where ρ is a path (a sequence of location names). The encoding of transactions requires an auxiliary encoding, denoted $\mathtt{D}\|\cdot\|^n_{\rho}$, loosely related to component (b) in (2). In case of an abortion signal $\overline{t}$, $\mathtt{D}\|\cdot\|^n_{\rho}$ defines a process that collects the encodings of the n protected blocks included in the default activity (which is to be found at ρ) as well as the encoding of the compensation activity. We define $\mathtt{D}\|\cdot\|^n_{\rho}$ by induction on n:

Definition 4.3 (Auxiliary Encoding)**.** *Let Q be a compensable process and let $\rho_0 = t,\rho$ be a path. Also, let $n \geq 0$. The process $\mathtt{D}\|Q\|^n_{\rho_0}$ is defined as follows:*

$$\begin{array}{rl}\mathtt{D}\|Q\|^0_{t,\rho} = & l_t.\overline{m_t}.p_\rho\left[\mathtt{D}[\![Q]\!]_\varepsilon\right] \mid m_t.\overline{k_t}.t\{\dagger\}\\ \mathtt{D}\|Q\|^n_{t,\rho} = & l_t.p_{t,\rho}\Big\{(X_1,\cdots,X_n).z\big\{p_\rho[X_1] \mid \cdots \mid p_\rho[X_n] \mid \overline{m_t}.p_\rho\left[\mathtt{D}[\![Q]\!]_\varepsilon\right]\big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \quad [n>0]\end{array}$$

(The definition of $\mathtt{D}[\![\cdot]\!]_\rho$ is given next.) Consider the encoding of $t[P,Q]$: if P contains n top-level protected blocks, then process $\mathtt{D}[\![t[P,Q]]\!]_\rho$ will include n successive update prefixes that will look for n protected blocks at location $p_{t,\rho}$ (the path points that they were enclosed with t) and move them to their parent location p_ρ. As these n dynamic updates leave these located processes at location t, an update on z is introduced to take them out of t once the n updates are executed.

We are now ready to introduce the encoding $\mathtt{D}[\![\cdot]\!]_\rho$. Recall that we adhere to Remark 4.1:

Definition 4.4 (Encoding Discarding Semantics)**.** *Let P be a compensable process and let ρ be a path. The encoding $\mathtt{D}[\![\cdot]\!]_\rho$ of compensable processes into adaptable processes is defined as follows:*

$$\mathtt{D}[\![\langle P\rangle]\!]_\rho = p_\rho\left[\mathtt{D}[\![P]\!]_\varepsilon\right] \qquad \mathtt{D}[\![t[P,Q]]\!]_\rho = t\left[\mathtt{D}[\![P]\!]_{t,\rho}\right] \mid \mathtt{D}\|Q\|^{\mathtt{npb}_\mathtt{D}(P)}_{t,\rho} \mid t.\overline{l_t}.k_t.\mathbf{0} \qquad \mathtt{D}[\![\mathbf{0}]\!]_\rho = \mathbf{0}$$

$$\mathtt{D}[\![P_1 \mid P_2]\!]_\rho = \mathtt{D}[\![P_1]\!]_\rho \mid \mathtt{D}[\![P_2]\!]_\rho \quad \mathtt{D}[\![\pi.P]\!]_\rho = \pi.\mathtt{D}[\![P]\!]_\rho \quad \mathtt{D}[\![!P]\!]_\rho = !\,\mathtt{D}[\![P]\!]_\rho \quad \mathtt{D}[\![(\nu a)P]\!]_\rho = (\nu a)\mathtt{D}[\![P]\!]_\rho$$

Key cases are encodings of protected blocks and transactions, as motivated earlier. Each protected block is associated with a location p indexed with the path to the protected block. A transaction is encoded as the composition of three processes. The leftmost component encodes the default activity P preserving the nested structure. In case of an abortion signal on t, the rightmost component will execute the middle component by sending message $\overline{l_t}$. As already explained, this second component will find all the top-level encodings of protected blocks of P, moving them to locations p_ρ together with the encoding of compensation activity Q. We may formalize these observations using the following lemma:

Lemma 4.5. *Let $t[P,Q]$ be a transaction with default activity P and compensation Q. Then we have:*

$$t\left[\mathtt{D}[\![P]\!]_{t,\rho}\right] \mid \mathtt{D}\|Q\|^{\mathtt{npb}_\mathtt{D}(P)}_{t,\rho} \mid \overline{l_t}.k_t \quad \to^* \quad \mathtt{D}[\![\mathtt{extr}_\mathtt{D}(P)]\!]_\rho \mid \mathtt{D}[\![\langle Q\rangle]\!]_\rho.$$

The following statement attests the operational correspondence for our encoding:

Theorem 4.6. *Let P be a compensable process and let ρ be an arbitrary path.*

a) If $P \xrightarrow{\tau}_\mathtt{D} P'$ then $\mathtt{D}[\![P]\!]_\rho \to^* \mathtt{D}[\![P']\!]_\rho$.

b) If $\mathtt{D}[\![P]\!]_\rho \to Q$ then there is P' such that $P \xrightarrow{\tau}_\mathtt{D} P'$ and $Q \to^* \mathtt{D}[\![P']\!]_\rho$.

We illustrate our encoding by means of an example:

Example 4.7. *Let $P_0 = t[R \mid \langle P\rangle, Q] \mid \overline{t}$ be a compensable process with $\mathtt{npb}_\mathtt{D}(R) = 0$. Then $P_0 \xrightarrow{\tau}_\mathtt{D} \langle P\rangle \mid \langle Q\rangle$. By expanding Def. 4.4, we obtain (recall that we omit $\mathbf{0}$ whenever possible):*

$$\begin{array}{rl}\mathtt{D}[\![P_0]\!]_\varepsilon = & t\left[\mathtt{D}[\![R \mid \langle P\rangle]\!]_{t,\varepsilon}\right] \mid \mathtt{D}\|Q\|^1_{t,\varepsilon} \mid t.\overline{l_t}.k_t \mid \overline{t}\\ = & t\left[\mathtt{D}[\![R]\!]_{t,\varepsilon} \mid p_{t,\varepsilon}\left[\mathtt{D}[\![P]\!]_\varepsilon\right]\right] \mid l_t.p_{t,\varepsilon}\Big\{(X).z\big\{p_\varepsilon[X] \mid \overline{m_t}.p_\varepsilon\left[\mathtt{D}[\![Q]\!]_\varepsilon\right]\big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \mid t.\overline{l_t}.k_t \mid \overline{t}\\ \to^* & t\left[\mathtt{D}[\![R]\!]_{t,\varepsilon} \mid z\big\{p_\varepsilon[\mathtt{D}[\![P]\!]_\varepsilon] \mid \overline{m_t}.p_\varepsilon\left[\mathtt{D}[\![Q]\!]_\varepsilon\right]\big\}\right] \mid z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\} \mid k_t \to^* p_\varepsilon\left[\mathtt{D}[\![P]\!]_\varepsilon\right] \mid p_\varepsilon\left[\mathtt{D}[\![Q]\!]_\varepsilon\right]\\ = & \mathtt{D}[\![\langle P\rangle \mid \langle Q\rangle]\!]_\varepsilon\end{array}$$

4.2 Preserving Semantics

The encoding of compensable processes with preserving semantics is as the previous encoding. In this case, since the extraction function keeps both protected blocks and top-level transactions (cf. Fig. 2), our auxiliary encoding, denoted $\mathtt{P}\|\cdot\|_{\rho}^{n,m}$, has two parameters: n denotes protected blocks and m denotes top-level transactions. We count protected blocks using Def. 4.2; to count transactions we use the following:

Definition 4.8 (Number of transactions)**.** *Let P be a compensable process. The number of transactions which occur in P, denoted $\mathtt{nts}(P)$, is defined as follows:*

$$\mathtt{nts}(P) = \begin{cases} \mathtt{nts}(P_1)+1 & \text{if } P = t[P_1, Q_1] \\ \mathtt{nts}(P_1)+\mathtt{nts}(P_2) & \text{if } P = P_1 \mid P_2 \\ \mathtt{nts}(P_1) & \text{if } P = (\nu a)P_1 \\ 0 & \text{otherwise.} \end{cases}$$

The encoding of the transaction body P with location t that is nested in location β_ρ.
Before giving the encoding $\mathtt{P}[\![\cdot]\!]_\rho$, we define the auxiliary encoding $\mathtt{P}\|\cdot\|_{\rho}^{n,m}$, where ρ is a path, n is the number of protected blocks, and m is the number of transactions in the default activity.

Definition 4.9 (Auxiliary Encoding)**.** *Let Q be a compensable process and let $\rho_0 = t, \rho$ be a path. Also, let $n, m \geq 0$. The process $\mathtt{P}\|Q\|_{\rho_0}^{n,m}$ is defined as follows:*

$$\begin{array}{rcl}
\mathtt{P}\|Q\|_{t,\rho}^{0,0} & = & l_t.\overline{m_t}.a.p_\rho\left[\mathtt{P}[\![Q]\!]_\varepsilon\right] \mid m_t.\overline{k_t}.t\{\dagger\} \\
\mathtt{P}\|Q\|_{t,\rho}^{1,0} & = & l_t.p_{t,\rho}\Big\{(X_1).z\big\{a.p_\rho[X_1] \mid \overline{m_t}.p_\rho[\mathtt{P}[\![Q]\!]_\varepsilon]\big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \\
\mathtt{P}\|Q\|_{t,\rho}^{0,1} & = & l_t.\beta_{t,\rho}\Big\{(Y_1).z\big\{a.\beta_\rho\left[Y_1\right] \mid \overline{m_t}.p_\rho\left[\mathtt{P}[\![Q]\!]_\varepsilon\right]\big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \\
\mathtt{P}\|Q\|_{t,\rho}^{n,m} & = & l_t.p_{t,\rho}\Big\{(X_1,\cdots,X_n).\beta_{t,\rho}\Big\{(Y_1,\cdots,Y_m).z\big\{p_\rho[X_1] \mid \cdots \mid p_\rho[X_n] \\
 & & \quad \mid a.(\beta_\rho[Y_1] \mid \cdots \mid \beta_\rho[Y_m]) \mid \overline{m_t}.p_\rho\left[\mathtt{P}[\![Q]\!]_\varepsilon\right]\big\}\Big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \quad [n,m>0]
\end{array}$$

We may now define the encoding $\mathtt{P}[\![\cdot]\!]_\rho$:

Definition 4.10 (Encoding Preserving)**.** *Let P be a compensable process and let ρ be a path. The encoding $\mathtt{P}[\![\cdot]\!]_\rho$ of compensable processes into adaptable processes is defined as*

$$\mathtt{P}[\![\langle P\rangle]\!]_\rho = p_\rho\left[\mathtt{P}[\![P]\!]_\varepsilon\right] \qquad \mathtt{P}[\![t[P,Q]]\!]_\rho = \beta_\rho\left[t\left[\mathtt{P}[\![P]\!]_{t,\rho}\right] \mid \mathtt{P}\|Q\|_{t,\rho}^{\mathtt{npb}_\mathtt{P}(P),\mathtt{nts}(P)} \mid t.\overline{l_t}.k_t.\overline{j}\right] \mid j.\beta_\rho\{\mathsf{id}\}.\overline{a}$$

and as a homomorphism for the other operators.

The following lemma formalizes the execution of the encoding:

Lemma 4.11. *Let $t[P,Q]$ be a transaction with default activity P and compensation Q. Then we have:*

$$\beta_\rho\left[t\left[\mathtt{P}[\![P]\!]_{t,\rho}\right] \mid \mathtt{P}\|Q\|_{t,\rho}^{\mathtt{npb}_\mathtt{P}(P),\mathtt{nts}(P)} \mid \overline{l_t}.k_t.\overline{j}\right] \mid j.\beta_\rho\{\mathsf{id}\}.\overline{a} \to^* \mathtt{P}[\![\mathtt{extr}_\mathtt{P}(P)]\!]_\rho \mid \mathtt{P}[\![\langle Q\rangle]\!]_\rho.$$

We then have the following statement of operational correspondence:

Theorem 4.12. *Let P be a compensable process and let ρ an arbitrary path.*

a) If $P \xrightarrow{\tau}_\mathtt{P} P'$ then $\mathtt{P}[\![P]\!]_\rho \to^* \mathtt{P}[\![P']\!]_\rho$.

b) If $\mathtt{P}[\![P]\!]_\rho \to Q$ then there is P' such that $P \xrightarrow{\tau}_\mathtt{P} P'$ and $Q \to^* \mathtt{P}[\![P']\!]_\rho$.

Example 4.13. *Let P_0 be a compensable process in Example 4.7 with $R = t_1[P_1, Q_1]$ and $\mathtt{npb}_{\mathtt{P}}(P_1) = \mathtt{nts}(P_1) = 0$. In the preserving semantics we have: $P_0 \xrightarrow{\tau}_{\mathtt{P}} t_1[P_1, Q_1] \mid \langle P\rangle \mid \langle Q\rangle$. By expanding Def. 4.10, we obtain:*

$$
\begin{aligned}
\mathtt{P}[\![P_0]\!]_\varepsilon &= \beta_\varepsilon\Big[t\big[\mathtt{P}[\![t_1[P_1,Q_1] \mid \langle P\rangle]\!]_{t,\varepsilon}\big] \mid \mathtt{P}\|Q\|^{1,1}_{t,\varepsilon} \mid t.\overline{l_t}.k_t.\overline{j}\Big] \mid j.\beta_\varepsilon\{\mathsf{id}\}.\overline{a} \mid \overline{t} \\
&= \beta_\varepsilon\Big[t\big[\beta_{t,\varepsilon}\big[M\big] \mid j.\beta_{t,\varepsilon}\{\mathsf{id}\}.\overline{a} \mid p_{t,\varepsilon}[\mathtt{P}[\![P]\!]_\varepsilon]\big] \mid l_t.p_{t,\varepsilon}\Big\{(X_1).\beta_{t,\varepsilon}\Big\{(Y_1).z\big\{p_\varepsilon\big[X_1\big] \mid a.\beta_\varepsilon\big[Y_1\big] \\
&\qquad\qquad \mid \overline{m_t}.p_\varepsilon\big[\mathtt{P}[\![Q]\!]_\varepsilon\big]\big\}\Big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \mid t.\overline{l_t}.k_t.\overline{j}\Big] \mid j.\beta_\varepsilon\{\mathsf{id}\}.\overline{a} \mid \overline{t} \\
&\to^* \beta_\varepsilon\Big[t\big[z\big\{p_\varepsilon\big[\mathtt{P}[\![P]\!]_\varepsilon\big] \mid a.\beta_\varepsilon\big[M\big] \mid \overline{m_t}.p_\varepsilon\big[\mathtt{P}[\![Q]\!]_\varepsilon\big]\big\} \mid j.\beta_{t,\varepsilon}\{\mathsf{id}\}.\overline{a}\big] \mid z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\} \mid k_t.\overline{j}\Big] \\
&\quad \mid j.\beta_\varepsilon\{\mathsf{id}\}.\overline{a} \\
&\to^* \beta_\varepsilon\Big[t\big[\mathbf{0} \mid j.\beta_{t,\varepsilon}\{\mathsf{id}\}.\overline{a}\big] \mid p_\varepsilon\big[\mathtt{P}[\![P]\!]_\varepsilon\big] \mid a.\beta_\varepsilon\big[M\big] \mid p_\varepsilon\big[\mathtt{P}[\![Q]\!]_\varepsilon\big] \mid t\{\dagger\} \mid \overline{j}\big] \mid j.\beta_\varepsilon\{\mathsf{id}\}.\overline{a} \\
&\to^* \beta_\varepsilon\big[M\big] \mid p_\varepsilon\big[\mathtt{P}[\![P]\!]_\varepsilon\big] \mid p_\varepsilon\big[\mathtt{P}[\![Q]\!]_\varepsilon\big]
\end{aligned}
$$

where $M = t_1\big[\mathtt{P}[\![P_1]\!]_{t_1,t,\varepsilon}\big] \mid \mathtt{P}\|Q_1\|^{0,0}_{t_1,t,\varepsilon} \mid t_1.\overline{l_{t_1}}.k_{t_1}.\overline{j}$.

4.3 Aborting Semantics

We now discuss the encoding of compensable processes with abortion semantics. While preserving the structure of the two encodings already presented, in this case the extraction function (cf. Fig. 2) add some complications. We need to modify the function that counts the number of protected blocks in a process; also, collecting encodings of (nested) protected blocks requires so-called *activation processes* which capture the hierarchical structure of nested transactions (cf. Def. 4.16).

Definition 4.14 (Number of protected blocks)**.** *Let P be a compensable process. The number of protected blocks in P, denoted by* $\mathtt{npb}_{\mathtt{A}}(P)$*, is defined as follows:*

$$
\mathtt{npb}_{\mathtt{A}}(P) = \begin{cases} 1 & \text{if } P = \langle P_1\rangle \\ \mathtt{npb}_{\mathtt{A}}(P_1)+1 & \text{if } P = t[P_1,Q_1] \\ \mathtt{npb}_{\mathtt{A}}(P_1)+\mathtt{npb}_{\mathtt{A}}(P_2) & \text{if } P = P_1 \mid P_2 \\ \mathtt{npb}_{\mathtt{A}}(P_1) & \text{if } P = (\nu a)P_1 \\ 0 & \text{otherwise.} \end{cases}
$$

We now define the auxiliary encoding, denoted $\mathtt{A}\|Q\|^n_\rho$. This process, as explained above, collects all encoded protected blocks of a process, in a case that an error notification is activated.

Definition 4.15 (Auxiliary Encoding)**.** *Let Q be a compensable process and let $\rho_0 = t,\rho$ be a path. Also, let $n \geq 0$. The process $\mathtt{A}\|Q\|^n_{\rho_0}$ is defined as follows:*

$$
\begin{aligned}
&\mathtt{A}\|Q\|^0_{t,\rho} = l_t.\overline{m_t}.p_\rho\big[\mathtt{A}[\![Q]\!]_\varepsilon\big] \mid m_t.\overline{k_t}.t\{\dagger\}.\Gamma_{t,\rho} \\
&\mathtt{A}\|Q\|^n_{t,\rho} = l_t.p_{t,\rho}\Big\{(X_1,\cdots,X_n).z\big\{p_\rho[X_1] \mid \cdots \mid p_\rho[X_n] \mid \overline{m_t}.p_\rho\big[\mathtt{A}[\![Q]\!]_\varepsilon\big]\big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}.\Gamma_{t,\rho})\ [n>0]
\end{aligned}
$$

where $\Gamma_{t,\rho} = \gamma_{t_1}\{(Z_1).\gamma_{t_1}[(\nu l_t)(\nu k_t)(Z_1 \mid l_t.\overline{k_t})]\}.\cdots.\gamma_{t_n}\{(Z_n).\gamma_{t_n}[(\nu l_t)(\nu k_t)(Z_n \mid l_t.\overline{k_t})]\}.\gamma_t\{\dagger\}$.

To appropriately collect nested protected blocks, we define a so-called *activation process* that captures the hierarchical structure of nested transactions.

Definition 4.16 (Activation Process)**.** *Let $St(P)$ denote the* containment structure *of compensable process P, i.e., the labeled tree (with root t) in which nodes are labeled with transaction names and sub-trees capture transaction nesting. The* activation process *for P, denoted $\mathscr{T}_t(P)$, is the sequential process obtained by a post-order search in $St(P)$ in which the visit to a node labeled c_i adds prefixes $\overline{l_{c_i}}.k_{c_i}$.*

This way, e.g., given $P = a\big[c[P_1, Q_2] \mid P_2, Q_1\big] \mid b\big[P_3 \mid d\big[P_4, Q_4\big] \mid e\big[P_5, Q_5\big], Q_3\big]$ we will have the activation process $\mathscr{T}_t(P) = \overline{l_c}.k_c.\overline{l_a}.k_a.\overline{l_d}.k_d.\overline{l_e}.k_e.\overline{l_b}.k_b.\overline{l_t}.k_t$.

Now we have all necessary definitions for introducing of the encoding $\mathtt{A}[\![\cdot]\!]_\rho$ of compensable processes into adaptable processes with respect to aborting semantics. Notice the use of activation processes in the encoding of transactions:

Definition 4.17 (Encoding Aborting)**.** *Let P be a compensable process and let ρ be a path. The encoding $\mathtt{A}[\![\cdot]\!]_\rho$ of compensable processes into adaptable processes is defined as*

$$\mathtt{A}[\![\langle P\rangle]\!]_\rho \;=\; p_\rho\big[\mathtt{A}[\![P]\!]_\varepsilon\big] \qquad\qquad \mathtt{A}[\![t[P,Q]]\!]_\rho = t\big[\mathtt{A}[\![P]\!]_{t,\rho}\big] \mid \mathtt{A}\|Q\|_{t,\rho}^{\mathtt{npb}_\mathtt{A}(P)} \mid \gamma_t\big[t.\mathscr{T}_t(P)\big]$$

and as a homomorphism for the other operators.

With respect to previous encodings, the encoding for aborting semantics differs in the rightmost process of the encoding. In this case, the activation process $\mathscr{T}_t(P)$ searches the subtree of the transaction body to activate the middle components of all nested transactions inside t.

The following correctness statements follow the same ideas as in the two previous encodings. In the sequel, we write $\approx$ to denote a (weak) behavioral equivalence that abstracts from internal transitions (due to the synchronizations added by the activation process).

Lemma 4.18. *Let $t[P,Q]$ be a transaction with default activity P and compensation Q. Then we have:*

$$t\big[\mathtt{A}[\![P]\!]_{t,\rho}\big] \mid \mathtt{A}\|Q\|_{t,\rho}^{\mathtt{npb}_\mathtt{A}(P)} \mid \gamma_t\big[\mathscr{T}_t(P)\big] \to^* \mathtt{A}[\![\mathtt{extr}_\mathtt{A}(P)]\!]_\rho \mid \mathtt{A}[\![\langle Q\rangle]\!]_\rho \mid \Gamma_{t,\rho} \mid \gamma_t\big[\mathbf{0}\big].$$

Theorem 4.19. *Let P be a compensable process and let ρ be an arbitrary path.*

a) If $P \xrightarrow{\tau}_\mathtt{A} P'$ then $\mathtt{A}[\![P]\!]_\rho \to^* \mathtt{A}[\![P']\!]_\rho$.

b) If $\mathtt{A}[\![P]\!]_\rho \to Q$ then there is P' such that $P \xrightarrow{\tau}_\mathtt{A} P'$ and $Q \to^* Q'$ and $Q' \approx \mathtt{A}[\![P']\!]_\rho$.

5 Encoding Dynamic Compensation Processes

We discuss how to extend the previous encodings to account for compensation updates $\mathtt{inst}\lfloor\lambda Y.R\rfloor.P$. Due to space constraints, we only describe required extensions to previously given definitions/statements.

Discarding Semantics. We first have the following extension to Def. 4.2:

Definition 5.1 (Number of protected blocks)**.** *Let P be a compensable process such that $P = \mathtt{inst}\lfloor\lambda Y.R\rfloor.P_1$. The number of protected blocks in P, denoted by $\mathtt{npb}(P)$, is equal to $\mathtt{npb}(P_1)$.*

The definition of the auxiliary encoding, given in Def. 4.3, is extended as follows:

Definition 5.2 (Auxiliary encoding)**.** *Let Q be a compensable process and let $\rho_0 = t,\rho$ be a path. Also, let $n \geq 0$. The process ${}^{\mathtt{d}}_{\mathtt{D}}\|Q\|^n_{\rho_0}$ is defined inductively on n as follows:*

$$\begin{array}{rcl}
{}^{\mathtt{d}}_{\mathtt{D}}\|Q\|^0_{t,\rho} & = & l_t.\overline{m_t}.p_\rho\big[u[\overline{f}.\overline{g}]\big] \mid m_t.\overline{k_t}.t\{\dagger\} \mid v[u\{(Z).(Z \mid v_1[\mathtt{D}[\![Q]\!]_\varepsilon] \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}] \\
{}^{\mathtt{d}}_{\mathtt{D}}\|Q\|^n_{t,\rho} & = & l_t.p_{t,\rho}\Big\{(X_1,\cdots,X_n).z\big\{p_\rho[X_1] \mid \cdots \mid p_\rho[X_n] \mid \overline{m_t}.p_\rho\big[u[\overline{f}.\overline{g}]\big]\big\}\Big\} \\
& & \qquad .(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \mid v\big[u\{(Z).(Z \mid v_1[\mathtt{D}[\![Q]\!]_\varepsilon] \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}\big] \quad [n>0]
\end{array}$$

Based on the above modifications, the encoding of processes with dynamic compensations is obtained by extending Def. 4.4 with the following:

$$\begin{array}{rcl} \mathtt{D}[\![Y]\!]_\rho & = & Y \\ \mathtt{D}[\![\mathtt{inst}\lfloor\lambda Y.R\rfloor.P]\!]_{t,\rho} & = & u[\mathbf{0}] \mid v_1\{(Y).\overline{g}.v\{(X).X \mid v[u\{(Z).(Z \mid \\ & & \qquad v_1[\mathtt{D}[\![R]\!]_\rho] \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}]\} \mid \mathtt{D}[\![P]\!]_{t,\rho}\}.\overline{f}.(v[\mathbf{0}] \mid v_1[\mathbf{0}]) \end{array}$$

We then have the following property:

Lemma 5.3. *Let $t[P,Q]$ be a transaction with default activity P and compensation Q. Then we have:*

$$t\big[\mathtt{D}[\![P]\!]_\rho\big] \mid {}^{\mathtt{d}}_{\mathtt{D}}\|Q\|_{t,\rho}^{\mathtt{npb}_\mathtt{D}(P)} \mid \overline{l_t}.k_t \to^* \mathtt{D}[\![\mathsf{extr}_\mathtt{D}(P)]\!]_\rho \mid \mathtt{D}[\![\langle P\rangle]\!]_\rho$$

Lemma 5.4. *If R is a compensable process such that all free occurrences of process variable X in it are replaced with a process Q then the following encoding holds: $[\![R\{^Q/_X\}]\!]_\rho = [\![R]\!]_\rho\{^{[\![Q]\!]_\rho}/_X\}$.*

Operational correspondence for the extended encoding follows from the following theorem:

Theorem 5.5. *Let P be a compensable process and let ρ be an arbitrary path.*

a) If $P \xrightarrow{\tau}_\mathtt{D} P'$ then there is an adaptable process P'' such that $\mathtt{D}[\![P]\!]_\rho \to^* P''$ and $P'' \approx \mathtt{D}[\![P']\!]_\rho$.

b) If $\mathtt{D}[\![P]\!]_\rho \to Q$ then there is P' such that $P \xrightarrow{\tau}_\mathtt{D} P'$ and $Q \to^* \mathtt{D}[\![P']\!]_\rho$.

Preserving Semantics. The function that counts the number of protected blocks in $\mathtt{inst}\lfloor\lambda Y.R\rfloor.P$ is the same as in Def. 5.1, while a function that counts the number of transactions is defined next.

Definition 5.6 (Number of transactions)**.** *Let P be a compensable process such that $P = \mathtt{inst}\lfloor\lambda Y.R\rfloor.P_1$. The number of transactions which occur in P, denoted $\mathtt{nts}(P)$, is equal to $\mathtt{nts}(P_1)$.*

We have the following extension of Def. 4.9:

Definition 5.7 (Auxiliary encoding)**.** *Let Q be a compensable process and let $\rho_0 = t,\rho$ be a path. Also, let $n,m \geq 0$. The process ${}^{\mathtt{d}}_{\mathtt{P}}\|Q\|_{\rho_0}^{n,m}$ is defined as follows:*

$$\begin{array}{rcl} {}^{\mathtt{d}}_{\mathtt{P}}\|Q\|_{t,\rho}^{0,0} & = & l_t.\overline{m_t}.a.p_\rho\big[u[\overline{f}.\overline{g}]\big] \mid m_t.\overline{k_t}.t\{\dagger\} \mid v\big[u\{(Z).(Z \mid v_1[\mathtt{P}[\![Q]\!]_\varepsilon] \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}\big] \\ {}^{\mathtt{d}}_{\mathtt{P}}\|Q\|_{t,\rho}^{1,0} & = & l_t.p_{t,\rho}\Big\{(X_1).z\big\{a.p_\rho[X_1] \mid \overline{m_t}.p_\rho[u[\overline{f}.\overline{g}]]\big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \\ & & \mid v[u\{(Z).(Z \mid v_1[\mathtt{P}[\![Q]\!]_\varepsilon] \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}] \\ {}^{\mathtt{d}}_{\mathtt{P}}\|Q\|_{t,\rho}^{0,1} & = & l_t.\beta_{t,\rho}\Big\{(Y_1).z\big\{a.\beta_\rho\big[Y_1\big] \mid \overline{m_t}.p_\rho\big[u[\overline{f}.\overline{g}]\big]\big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \\ & & \mid v\big[u\{(Z).(Z \mid v_1[\mathtt{P}[\![Q]\!]_\varepsilon] \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}\big] \\ {}^{\mathtt{d}}_{\mathtt{P}}\|Q\|_{t,\rho}^{n,m} & = & l_t.p_{t,\rho}\Big\{(X_1,\cdots,X_n).\beta_{t,\rho}\Big\{(Y_1,\cdots,Y_m).z\big\{p_\rho[X_1] \mid p_\rho[X_2] \mid \cdots \mid p_\rho[X_n] \\ & & \mid a.(\beta_\rho[Y_1] \mid \cdots \mid \beta_\rho[Y_m]) \mid \overline{m_t}.p_\rho\big[u[\overline{f}.\overline{g}]\big]\big\}\Big\}.(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}) \mid v\big[u\{(Z).(Z \mid v_1[\mathtt{P}[\![Q]\!]_\varepsilon] \\ & & \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}\big] \qquad [n,m>0] \end{array}$$

We then have the following extended correctness statements:

Lemma 5.8. *Let $t[P,Q]$ be a transaction with default activity P and compensation Q. Then we have:*

$$\beta_\rho\Big[t\big[\mathtt{P}[\![P]\!]_{t,\rho}\big] \mid {}^{\mathtt{d}}_{\mathtt{P}}\|Q\|_{t,\rho}^{\mathtt{npb}_\mathtt{P}(P),\mathtt{nts}(P)} \mid \overline{l_t}.k_t.\overline{j}\Big] \mid j.\beta_\rho\{\mathsf{id}\}.\overline{a} \to^* \mathtt{P}[\![\mathsf{extr}_\mathtt{P}(P)]\!]_\rho \mid \mathtt{P}[\![\langle Q\rangle]\!]_\rho$$

Theorem 5.9. *Let P be a compensable process and let ρ be an arbitrary path.*

a) If $P \xrightarrow{\tau}_\mathtt{P} P'$ then there is an adaptable process P'' such that $\mathtt{P}[\![P]\!]_\rho \to^* P''$ and $P'' \approx \mathtt{P}[\![P']\!]_\rho$.

b) If $\mathtt{P}[\![P]\!]_\rho \to Q$ then there is P' such that $P \xrightarrow{\tau}_\mathtt{P} P'$ and $Q \to^* \mathtt{P}[\![P']\!]_\rho$.

Aborting Semantics. The encoding of processes with dynamic compensations and aborting semantics is obtained by extending Def. 4.17 with the encodings of process variables and compensation updates, which are the same as in discarding and preserving semantics. The function that counts protected blocks in compensation updates $\mathtt{npb}(\mathtt{inst}\lfloor\lambda Y.R\rfloor.P)$ is as in Def. 5.1. We require an extension to Def. 4.15:

Definition 5.10 (Auxiliary encoding)**.** *Let Q be a compensable process and let $\rho_0 = t,\rho$ be a path. Also, let $n \geq 0$. The process $\overset{\mathtt{d}}{\mathtt{A}}\|Q\|^n_{\rho_0}$ is defined as follows:*

$$\begin{array}{rcl}
\overset{\mathtt{d}}{\mathtt{A}}\|Q\|^0_{t,\rho} & = & l_t.\overline{m_t}.p_\rho\left[u[\overline{f}.\overline{g}]\right] \mid m_t.\overline{k_t}.t\{\dagger\}.\Gamma_{t,\rho} \mid v[u\{(Z).(Z \mid v_1[\mathtt{A}[\![Q]\!]_\varepsilon] \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}] \\
\overset{\mathtt{d}}{\mathtt{A}}\|Q\|^n_{t,\rho} & = & l_t.p_{t,\rho}\Big\{(X_1,\cdots,X_n).z\{p_\rho[X_1] \mid p_\rho[X_2] \mid \cdots \mid p_\rho[X_n] \mid \overline{m_t}.p_\rho\left[u[\overline{f}.\overline{g}]\right]\}\Big\} \\
& & \quad .(z[\mathbf{0}] \mid m_t.\overline{k_t}.t\{\dagger\}.\Gamma_{t,\rho}) \mid v[u\{(Z).(Z \mid v_1[\mathtt{A}[\![Q]\!]_\varepsilon] \mid f.v_1\{\mathsf{id}\}.v\{\mathsf{id}\}.g)\}] \quad [n>0]
\end{array}$$

We then have the following extended correctness statements:

Lemma 5.11. *Let $t[P,Q]$ be a transaction with default activity P and compensation Q. We have:*

$$t\left[\mathtt{A}[\![P]\!]_{t,\rho}\right] \mid \overset{\mathtt{d}}{\mathtt{A}}\|Q\|^{\mathtt{npb}_\mathtt{A}(P)}_{t,\rho} \mid \gamma_t\left[\mathscr{T}_t(P)\right] \rightarrow^* \mathtt{A}[\![\mathsf{extr}_\mathtt{A}(P)]\!]_\rho \mid \mathtt{A}[\![\langle Q\rangle]\!]_\rho \mid \Gamma_{t,\rho} \mid \gamma_t\left[\mathbf{0}\right].$$

Theorem 5.12. *Let P be a compensable process and let ρ be an arbitrary path.*

a) If $P \xrightarrow{\tau}_\mathtt{A} P'$ then there is an adaptable process P'' such that $\mathtt{A}[\![P]\!]_\rho \rightarrow^* P''$ and $P'' \approx \mathtt{A}[\![P']\!]_\rho$.

b) If $\mathtt{A}[\![P]\!]_\rho \rightarrow Q$ then there is P' such that $P \xrightarrow{\tau}_\mathtt{A} P'$ and $Q \rightarrow^* \mathtt{A}[\![P']\!]_\rho$.

6 Concluding Remarks

We have compared, from the point of view of relative expressiveness, two related and yet fundamentally different process models: the calculus of *compensable processes* (introduced in [12]) and the calculus of *adaptable processes* (introduced in [2]). We provided encodings of processes with static and dynamic compensations (under discarding, preserving, and aborting semantics) into adaptable processes. Our encodings not only are a non trivial application of process mobility as present in adaptable processes; they also shed light on the intricate semantics of compensable processes. As encoding criteria, we have considered compositionality and operational correspondence (up-to weak equivalences), as in [11]. It would be insightful to establish encoding correctness with respect to all the criteria in [11].

Our study opens several interesting avenues for future work. Having addressed the encodability of compensable processes into adaptable processes, we plan to consider the reverse direction, i.e., encodings of adaptable processes into compensable processes. We conjecture that an encoding of adaptable process into a language with static compensations does not exist: compensation updates $\mathtt{inst}\lfloor\lambda X.Q\rfloor.P$ seem essential to model an update prefix $l\{(X).Q\}.P$—the semantics of both constructs induces process substitutions. Still, even by considering a language with dynamic compensations, an encoding of adaptable processes is far from obvious, because the semantics of compensation updates dynamically modifies the behavior of the compensation activity, the inactive part of a transaction. Formalizing these (non) encodability claims is interesting future work. Another promising direction is to cast our encodability results into variants of adaptable and compensable processes with *session types*: a candidate for source language could be the typed calculus with *interactional exceptions* developed in [6]; as target language, we plan to consider extensions of adaptable processes with session types [9, 10].

Acknowledgements. We are grateful to the anonymous reviewers for their comments and suggestions. This research was partially supported by the EU COST Action IC1201. Pérez is also affiliated to NOVA Laboratory for Computer Science and Informatics, Universidade Nova de Lisboa, Portugal.

References

[1] Laura Bocchi, Cosimo Laneve & Gianluigi Zavattaro (2003): *A Calculus for Long-Running Transactions*. In: *Proc. of FMOODS 2003, LNCS* 2884, Springer, pp. 124–138, doi:10.1007/978-3-540-39958-2_9.

[2] Mario Bravetti, Cinzia Di Giusto, Jorge A. Pérez & Gianluigi Zavattaro (2012): *Adaptable processes. Logical Methods in Computer Science* 8(4), doi:10.2168/LMCS-8(4:13)2012.

[3] Mario Bravetti, Cinzia Di Giusto, Jorge A. Pérez & Gianluigi Zavattaro (2012): *Towards the Verification of Adaptable Processes*. In: *ISoLA, LNCS* 7609, Springer, pp. 269–283, doi:10.1007/978-3-642-34026-0_20.

[4] Mario Bravetti & Gianluigi Zavattaro (2009): *On the expressive power of process interruption and compensation*. *Mathematical Structures in Computer Science* 19(3), pp. 565–599, doi:10.1017/S0960129509007683.

[5] Luís Caires, Carla Ferreira & Hugo Torres Vieira (2009): *A Process Calculus Analysis of Compensations*. In: *Proc. of TGC 2008, LNCS* 5474, Springer, pp. 87–103, doi:10.1007/978-3-642-00945-7_6.

[6] Marco Carbone, Kohei Honda & Nobuko Yoshida (2008): *Structured Interactional Exceptions in Session Types*. In: *CONCUR 2008, LNCS* 5201, Springer, pp. 402–417, doi:10.1007/978-3-540-85361-9_32.

[7] Jovana Dedeić, Jovanka Pantović & Jorge A. Pérez (2015): *Full version of this paper*. Technical Report. Available at http://www.jorgeaperez.net.

[8] Carla Ferreira, Ivan Lanese, António Ravara, Hugo Torres Vieira & Gianluigi Zavattaro (2011): *Advanced Mechanisms for Service Combination and Transactions*. In: *Results of SENSORIA, LNCS* 6582, Springer, pp. 302–325, doi:10.1007/978-3-642-20401-2_14.

[9] Cinzia Di Giusto & Jorge A. Pérez (2015): *Disciplined structured communications with disciplined runtime adaptation*. *Sci. Comput. Program.* 97, pp. 235–265, doi:10.1016/j.scico.2014.04.017.

[10] Cinzia Di Giusto & Jorge A. Pérez (2015): *An Event-Based Approach to Runtime Adaptation in Communication-Centric Systems*. In: *Proc. of WS-FM 2014*, LNCS, Springer. To appear.

[11] Daniele Gorla (2010): *Towards a unified approach to encodability and separation results for process calculi*. *Inf. Comput.* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[12] Ivan Lanese, Cátia Vaz & Carla Ferreira (2010): *On the Expressive Power of Primitives for Compensation Handling*. In: *Proc. of ESOP 2010, LNCS* 6012, Springer, pp. 366–386, doi:10.1007/978-3-642-11957-6_20.

[13] Ivan Lanese & Gianluigi Zavattaro (2013): *Decidability Results for Dynamic Installation of Compensation Handlers*. In: *COORDINATION, LNCS* 7890, Springer, pp. 136–150, doi:10.1007/978-3-642-38493-6_10.

[14] Cosimo Laneve & Gianluigi Zavattaro (2005): *Foundations of Web Transactions*. In: *Proc. of FOSSACS 2005, LNCS* 3441, Springer, pp. 282–298, doi:10.1007/978-3-540-31982-5_18.

[15] Robin Milner (1989): *Communication and concurrency*. PHI Series in computer science, Prentice Hall.

[16] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, I*. *Inf. Comput.* 100(1), pp. 1–40, doi:10.1016/0890-5401(92)90008-4.

[17] Davide Sangiorgi (1992): *Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms*. Ph.D. thesis, University of Edinburgh.

SOS rule formats for convex and abstract probabilistic bisimulations*

Pedro R. D'Argenio Matias David Lee
FaMAF, Universidad Nacional de Córdoba – CONICET.
Ciudad Universitaria, X5000HUA – Córdoba, Argentina.
dargenio@famaf.unc.edu.ar lee@famaf.unc.edu.ar

Daniel Gebler
Department of Computer Science, VU University Amsterdam,
De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands
e.d.gebler@vu.nl

Probabilistic transition system specifications (PTSSs) in the $nt\mu f\theta/nt\mu x\theta$ format provide structural operational semantics for Segala-type systems that exhibit both probabilistic and nondeterministic behavior and guarantee that bisimilarity is a congruence for all operator defined in such format. Starting from the $nt\mu f\theta/nt\mu x\theta$, we obtain restricted formats that guarantee that three coarser bisimulation equivalences are congruences. We focus on (i) Segala's variant of bisimulation that considers combined transitions, which we call here *convex bisimulation*; (ii) the bisimulation equivalence resulting from considering Park & Milner's bisimulation on the usual stripped probabilistic transition system (translated into a labelled transition system), which we call here *probability obliterated bisimulation*; and (iii) a *probability abstracted bisimulation*, which, like bisimulation, preserves the structure of the distributions but instead, it ignores the probability values. In addition, we compare these bisimulation equivalences and provide a logic characterization for each of them.

1 Introduction

Structural operational semantics (SOS for short) [24] is a powerful tool to provide semantics to programming languages. In SOS, process behavior is described using transition systems and the behavior of a composite process is given in terms of the behavior of its components. SOS has been formalized using an algebraic framework as *Transition Systems Specifications (TSS)* [6, 7, 14, 15, 23, etc.]. Basically, a TSS contains a signature, a set of actions or labels, and a set of rules. The signature defines the terms in the language. The set of actions represents all possible activities that a process (i.e., a term over the signature) can perform. The rules define how a process should behave (i.e., perform certain activities) in terms of the behavior of its subprocesses, that is, the rules define compositionally the transition system associated to each term of the language. A particular focus of these formalizations was to provide a meta-theory that ensures a diversity of semantic properties by simple inspection on the form of the rules. (See [1, 2, 23] for overviews.) One of such kind of properties is to ensure that a given equivalence relation is a congruence for all operators whose semantics is defined in a TSS whose rules complies to a particular format. These so called *congruence theorems* have been proved for a variety of equivalences in the non-probabilistic case [6, 14, 15, etc.].

*Supported by ANPCyT project PICT-2012-1823, SeCyT-UNC program 05/BP12 and their related projects, and EU 7FP grant agreement 295261 (MEALS).

S. Crafa and D. Gebler (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2015)
EPTCS 190, 2015, pp. 31–45, doi:10.4204/EPTCS.190.3

© D'Argenio, Lee & Gebler
This work is licensed under the Creative Commons Attribution License.

The introduction of probabilistic process algebras motivated the need for a theory of structural operational semantics to define *probabilistic* transition systems. Few earlier results appeared in this direction [4, 5, 17, 18] presenting congruence theorems for Larsen & Skou bisimulation equivalence [19]. Most of these formats have complicated restrictions that extend to sets of rules due to the fact that they considered transitions labeled both with an action and a probability value. By using a more modern view of probabilistic transition systems (where the target of the transition is a probability distribution on states) we manage to obtained the most general format for bisimulation equivalence, which we called $nt\mu f\theta/nt\mu x\theta$, following the nomenclature of [14, 15].

Starting from the $nt\mu f\theta/nt\mu x\theta$ format, in this paper we define formats to guarantee that three coarser versions of bisimulation equivalence are congruences for all operator definable in the respective format. The first relation we focus on is Segala's variant of bisimulation that considers combined transitions, here called *convex bisimulation* [25]. The second relation we explore originates here and we call it *probability abstracted bisimulation*. Like bisimulation and unlike convex bisimulation, it preserves the structure of the distributions of each transition, but instead, it ignores the probability values. This relation preserves the fairness introduced by the probability distributions. Finally, we study the bisimulation equivalence resulting from considering Park & Milner's bisimulation [22] on the usual stripped probabilistic transition system (translated into a labeled transition system). Here we call it *probability obliterated bisimulation*. This is the usual way to abstract probabilities, but it has the drawback that it breaks the basic fairness provided by probabilistic choices.

Apart from presenting congruence theorems for all previously mentioned bisimulation equivalences, we briefly study alternative definitions of these bisimulations, compare them with each other, and provide logical characterizations, which are particularly new here for probability abstracted and probability obliterated bisimulation equivalences.

The paper is organized as follows. Sec. 2 recalls the type of algebraic structure and Sec. 3 provides the basic notions and results of probabilistic transition system specifications (PTSS). Sec. 4 presents the different bisimulation equivalences and a brief study of them, including their logical characterizations. The study of all the PTSS formats and the respective congruence theorems is given in Sec. 5. The paper concludes in Sec 6.

2 Preliminaries

Let $S = \{s, d\}$ be a set denoting two sorts. Elements of sort $s \in S$ are intended to represent states in the transition system, while elements of sort $d \in S$ will represent distributions over states. We let σ range over S. An *S-sorted signature* is a structure (F, ar), where (i) F is a set of *function names*, and (ii) $\mathsf{ar}\colon F \to (S^* \times S)$ is the *arity function*. The rank of $f \in F$ is the number of arguments of f, defined by $\mathsf{rk}(f) = n$ if $\mathsf{ar}(f) = \sigma_1 \dots \sigma_n \to \sigma$. (We write "$\sigma_1 \dots \sigma_n \to \sigma$" instead of "$(\sigma_1 \dots \sigma_n, \sigma)$" to highlight that function f maps to sort σ.) Function f is a *constant* if $\mathsf{rk}(f) = 0$. To simplify the presentation we will write an S-sorted signature (F, ar) as a pair of disjoint signatures (Σ_s, Σ_d) where Σ_s is the set of operations that map to s and Σ_d is the set of operations that map to d. Let $\mathcal{V}$ and $\mathcal{V}_d$ be two infinite sets of S-sorted variables where $\mathcal{V}, \mathcal{V}_d, F$ are all mutually disjoint. We use x, y, z (with possible sub- or super-scripts) to range over $\mathcal{V}$, μ, ν to range over $\mathcal{V}_d$ and ζ to range over $\mathcal{V} \cup \mathcal{V}_d$.

Definition 1. *Let Σ_s and Σ_d be two signatures as before and let $V \subseteq \mathcal{V}$ and $D \subseteq \mathcal{V}_d$. We simultaneously define the sets of* state terms $T(\Sigma_s, V, D)$ *and* distribution terms $T(\Sigma_d, V, D)$ *as the smallest sets satisfying: (i) $V \subseteq T(\Sigma_s, V, D)$; (ii) $D \subseteq T(\Sigma_d, V, D)$; (iii) $f(\xi_1, \cdots, \xi_{\mathsf{rk}(f)}) \in T(\Sigma_\sigma, V, D)$, if $\mathsf{ar}(f) = \sigma_1 \dots \sigma_n \to \sigma$ and $\xi_i \in T(\Sigma_{\sigma_i}, V, D)$.*

We let $\mathbb{T}(\Sigma) = T(\Sigma_s, \mathcal{V}, \mathcal{V}_d) \cup T(\Sigma_d, \mathcal{V}, \mathcal{V}_d)$ denote the set of all *open terms* and distinguish the sets $\mathbb{T}(\Sigma_s) = T(\Sigma_s, \mathcal{V}, \mathcal{V}_d)$ of *open state terms* and $\mathbb{T}(\Sigma_d) = T(\Sigma_d, \mathcal{V}, \mathcal{V}_d)$ of *open distribution terms*. Similarly, we let $\mathsf{T}(\Sigma) = T(\Sigma_s, \emptyset, \emptyset) \cup T(\Sigma_d, \emptyset, \emptyset)$ denote the set of all *closed terms* and distinguish the sets $\mathsf{T}(\Sigma_s) = T(\Sigma_s, \emptyset, \emptyset)$ of *closed state terms* and $\mathsf{T}(\Sigma_d) = T(\Sigma_d, \emptyset, \emptyset)$ of *closed distribution terms*. We let t, t', t_1,... range over state terms, θ, θ', θ_1,... range over distribution terms, and ξ, ξ', ξ_1,... range over any kind of terms. With $\mathcal{V}(\xi) \subseteq \mathcal{V} \cup \mathcal{V}_d$ we denote the set of variables occurring in term ξ.

Let $\Delta(\mathsf{T}(\Sigma_s))$ denote the set of all (discrete) probability distributions on $\mathsf{T}(\Sigma_s)$. We let π range over $\Delta(\mathsf{T}(\Sigma_s))$. For each $t \in \mathsf{T}(\Sigma_s)$, let $\delta_t \in \Delta(\mathsf{T}(\Sigma_s))$ denote the *Dirac distribution*, i.e., $\delta_t(t) = 1$ and $\delta_t(t') = 0$ if t and t' are not syntactically equal. For $X \subseteq \mathsf{T}(\Sigma_s)$ we define $\pi(X) = \sum_{t \in X} \pi(t)$. The convex combination $\sum_{i \in I} p_i \pi_i$ of a family $\{\pi_i\}_{i \in I}$ of probability distributions with $p_i \in (0, 1]$ and $\sum_{i \in I} p_i = 1$ is defined by $(\sum_{i \in I} p_i \pi_i)(t) = \sum_{i \in I} (p_i \pi_i(t))$.

The type of signatures we consider has a particular construction. We start from a signature Σ_s of functions mapping into sort s and construct the signature Σ_d of functions mapping into d as follows. For each $f \in F_s$ we include a function symbol $\boldsymbol{f} \in F_d$ with $\mathsf{ar}(\boldsymbol{f}) = d \ldots d \to d$ and $\mathsf{rk}(\boldsymbol{f}) = \mathsf{rk}(f)$. We call $\boldsymbol{f}$ the *probabilistic lifting* of f. (We use boldface fonts to indicate that a function in Σ_d is the probabilistic lifting of another in Σ_s.) Moreover Σ_d may include any of the following additional operators: (i) δ with arity $\mathsf{ar}(\delta) = s \to d$, and (ii) $\bigoplus_{i \in I} [p_i]\_$ with I being a finite or countable infinite index set, $\sum_{i \in I} p_i = 1$, $p_i \in (0, 1]$ for all $i \in I$, and $\mathsf{ar}\left(\bigoplus_{i \in I} [p_i]\_\right) = d^{|I|} \to d$. Notice that if I is countably infinite, $\bigoplus_{i \in I} [p_i]\_$ is an infinitary operator.

Operators δ and $\bigoplus_{i \in I} [p_i]\_$ are used to construct discrete probability functions of countable support: $\delta(t)$ is interpreted as a distribution that assigns probability 1 to the state term t and probability 0 to any other term t' (syntactically) different from t, and $\bigoplus_{i \in I} [p_i]\theta_i$ represents a distribution that weights with p_i the distribution represented by the term θ_i. Moreover, a probabilistically lifted operator $\boldsymbol{f}$ is interpreted by properly lifting the probabilities of the operands to terms composed with the operator f.

Formally, the algebra associated with a probabilistically lifted signature $\Sigma = (\Sigma_s, \Sigma_d)$ is defined as follows. For sort s, it is the freely generated algebraic structure $\mathsf{T}(\Sigma_s)$. For sort d, it is defined by the carrier $\Delta(\mathsf{T}(\Sigma_s))$ and the following interpretation: $[\![\delta(t)]\!] = \delta_t$ for all $t \in \mathsf{T}(\Sigma_s)$, $[\![\bigoplus_{i \in I} [p_i]\theta_i]\!] = \sum_{i \in I} p_i [\![\theta_i]\!]$ for $\{\theta_i \mid i \in I\} \subseteq \mathsf{T}(\Sigma_d)$, $[\![\boldsymbol{f}(\theta_1, \ldots, \theta_{\mathsf{rk}(f)})]\!](f(\xi_1, \ldots, \xi_{\mathsf{rk}(f)})) = \prod_{\sigma_i = s} [\![\theta_i]\!](\xi_i)$ if for all j s.t. $\sigma_j = d$, $\theta_j = \xi_j$, and $[\![\boldsymbol{f}(\theta_1, \ldots, \theta_{\mathsf{rk}(f)})]\!](f(\xi_1, \ldots, \xi_{\mathsf{rk}(f)})) = 0$ otherwise. Here it is assumed that $\prod \emptyset = 1$. Notice that in the semantics of a lifted function $\boldsymbol{f}$, the big product only considers the distributions related to the s-sorted positions in f, while the distribution terms corresponding to the d-sorted positions in f should match exactly to the parameters of $\boldsymbol{f}$.

A *substitution* ρ is a map $\mathcal{V} \cup \mathcal{V}_d \to \mathbb{T}(\Sigma)$ such that $\rho(x) \in \mathbb{T}(\Sigma_s)$, for all $x \in \mathcal{V}$, and $\rho(\mu) \in \mathbb{T}(\Sigma_d)$, for all $\mu \in \mathcal{V}_d$. A substitution is closed if it maps each variable to a closed term. A substitution extends to a mapping from terms to terms as usual.

Finally, we remark a general property of distribution terms: let $f \in \Sigma_s$ with $\mathsf{ar}(f) = \sigma_1 \ldots \sigma_n \to s$, and let $\sigma_j = s$; then $\boldsymbol{f} \in \Sigma_d$ is distributive w.r.t. $\oplus$ in the position j, i.e. $[\![\rho(\boldsymbol{f}(\ldots, \xi_{j-1}, \bigoplus_{i \in I} [p_i]\theta_i, \xi_{j+1}, \ldots))]\!] = [\![\rho(\bigoplus_{i \in I} [p_i]\boldsymbol{f}(\ldots, \xi_{j-1}, \theta_i, \xi_{j+1}, \ldots))]\!]$ for any closed substitution ρ. The proof follows from the definition of $[\![\_]\!]$. However, notice that $\boldsymbol{f}$ *does not* distribute w.r.t. $\oplus$ in a position k such that $\sigma_k = d$.

3 Probabilistic Transition System Specifications

A (probabilistic) transition relation prescribes which possible activity can be performed by a term in a signature. Such activity is described by the label of the action and a probability distribution on terms that indicates the probability to reach a particular new term. We will follow the probabilistic automata style

of probabilistic transitions [25] which are a generalization of the so-called reactive model [19].

Definition 2 (PTS)**.** *A* probabilistic labeled transition system *(PTS) is a triple* $(\mathsf{T}(\Sigma_s), A, \rightarrow)$*, where* $\Sigma = (\Sigma_s, \Sigma_d)$ *is a probabilistically lifted signature, A is a countable set of actions, and* $\rightarrow \subseteq \mathsf{T}(\Sigma_s) \times A \times \Delta(\mathsf{T}(\Sigma_s))$*, is a transition relation. We write* $t \xrightarrow{a} \pi$ *for* $(t, a, \pi) \in \rightarrow$.

Transition relations are usually defined by means of structured operational semantics in Plotkin's style [24]. For PTS, algebraic characterizations of this style were provided in [8, 9, 21] where the term *probabilistic transition system specification* was used and which we adopt in our paper.

Definition 3 (PTSS)**.** *A* probabilistic transition system specification *(PTSS) is a triple* $P = (\Sigma, A, R)$ *where* Σ *is a probabilistically lifted signature, A is a set of labels, and R is a set of rules of the form:*

$$\frac{\{t_k \xrightarrow{a_k} \theta_k \mid k \in K\} \cup \{t_l \xrightarrow{b_l} \mid l \in L\} \cup \{\theta_j(T_j) \bowtie_j q_j \mid j \in J\}}{t \xrightarrow{a} \theta}$$

where K, L, J *are index sets,* $t, t_k, t_l \in \mathbb{T}(\Sigma_s)$*,* $a, a_k, b_l \in A$*,* $T_j \subseteq \mathsf{T}(\Sigma_s)$*,* $\bowtie_j \in \{>, \geq, <, \leq\}$*,* $q_j \in [0,1]$ *and* $\theta_j, \theta_k, \theta \in \mathbb{T}(\Sigma_d)$.

Expressions of the form $t \xrightarrow{a} \theta$, $t \xrightarrow{a} \not\rightarrow$, and $\theta(T) \bowtie p$ are called *positive literal*, *negative literal*, and *quantitative literal*, respectively. For any rule $r \in R$, literals above the line are called *premises*, notation $\mathrm{prem}(r)$; the literal below the line is called *conclusion*, notation $\mathrm{conc}(r)$. We denote with $\mathrm{pprem}(r)$, $\mathrm{nprem}(r)$, and $\mathrm{qprem}(r)$ the sets of positive, negative, and quantitative premises of the rule r, respectively. In general, we allow the sets of positive, negative, and quantitative premises to be infinite.

Substitutions provide instances to the rules of a PTSS that, together with some appropriate machinery, allow us to define probabilistic transition relations. Given a substitution ρ, it extends to literals as follows: $\rho(t \xrightarrow{a} \not\rightarrow) = \rho(t) \xrightarrow{a} \not\rightarrow$, $\rho(\theta(T) \bowtie p) = \rho(\theta)(\rho(T)) \bowtie p$ (where $\rho(T) = \{\rho(t) \mid t \in T\}$), and $\rho(t \xrightarrow{a} \theta) = \rho(t) \xrightarrow{a} \rho(\theta)$.

We say that r' is a (closed) instance of a rule r if there is a (closed) substitution ρ so that $r' = \rho(r)$. We say that ρ is a *proper substitution of* r if for all quantitative premises $\theta(T) \bowtie p$ of r and all $t \in T$, $[\![\rho(\theta)]\!](\rho(t)) > 0$ holds. We use only this kind of substitution in the paper.

In the rest of the paper, we will deal with models as *symbolic* transition relations in the set $\mathsf{T}(\Sigma_s) \times A \times \mathsf{T}(\Sigma_d)$ rather than the *concrete* transition relations in $\mathsf{T}(\Sigma_s) \times A \times \Delta(\mathsf{T}(\Sigma_s))$ required by a PTS. Hence we will mostly refer with the term "transition relation" to the symbolic transition relation. In any case, a symbolic transition relation induces always a unique concrete transition relation by interpreting every target distribution term as the distribution it defines; that is, the symbolic transition $t \xrightarrow{a} \theta$ is interpreted as the concrete transition $t \xrightarrow{a} [\![\theta]\!]$. If the symbolic transition relation turns out to be a model of a PTSS P, we say that the induced concrete transition relation defines a PTS associated to P.

To define an appropriate notion of model we consider *3-valued models*. A 3-valued model partitions the set $\mathsf{T}(\Sigma_s) \times A \times \mathsf{T}(\Sigma_d)$ in three sets containing, respectively, the transition that are known to hold, that are known not to hold, and those whose validity is unknown. Thus, a 3-valued model can be presented as a pair $\langle \mathsf{CT}, \mathsf{PT} \rangle$ of transition relations $\mathsf{CT}, \mathsf{PT} \subseteq \mathsf{T}(\Sigma_s) \times A \times \mathsf{T}(\Sigma_d)$, with $\mathsf{CT} \subseteq \mathsf{PT}$, where CT is the set of transitions that *certainly* hold and PT is the set of transitions that *possibly* hold. So, transitions in $\mathsf{PT} \setminus \mathsf{CT}$ are those whose validity is unknown and transitions in $(\mathsf{T}(\Sigma_s) \times A \times \mathsf{T}(\Sigma_d)) \setminus \mathsf{PT}$ are those that certainly do not hold. A 3-valued model $\langle \mathsf{CT}, \mathsf{PT} \rangle$ that is justifiably compatible with the proof system defined by a PTSS P is said to be *stable* for P. (See Def. 5.)

Before formally defining the notions of proof and 3-valued stable model we introduce some notation. Given a transition relation $\mathsf{Tr} \subseteq \mathsf{T}(\Sigma_s) \times A \times \mathsf{T}(\Sigma_d)$, $t \xrightarrow{a} \theta$ *holds in* Tr, notation $\mathsf{Tr} \models t \xrightarrow{a} \theta$, if $t \xrightarrow{a} \theta \in \mathsf{Tr}$; $t \xrightarrow{a} \not\rightarrow$ *holds in* Tr, notation $\mathsf{Tr} \models t \xrightarrow{a} \not\rightarrow$, if for all $\theta \in \mathsf{T}(\Sigma_d)$, $t \xrightarrow{a} \theta \notin \mathsf{Tr}$. A closed quantitative constraint

$\theta(T) \bowtie p$ *holds in* Tr, notation $\mathsf{Tr} \models \theta(T) \bowtie p$, if $[\![\theta]\!](T) \bowtie p$. Notice that the satisfaction of a quantitative constraint does not depend on the transition relation. We nonetheless use this last notation as it turns out to be convenient. Given a set of literals H, we write $\mathsf{Tr} \models H$ if for all $\phi \in H$, $\mathsf{Tr} \models \phi$.

Definition 4 (Proof)**.** *Let $P = (\Sigma, A, R)$ be a PTSS. Let ψ be a positive literal and let H be a set of literals. A* proof *of a transition rule $\frac{H}{\psi}$ from P is a well-founded, upwardly branching tree where each node is a literal such that: (i) the root is ψ; and (ii) if χ is a node and K is the set of nodes directly above χ, then one of the following conditions holds: (a) $K = \emptyset$ and $\chi \in H$, or (b) $\chi = (\theta(T) \bowtie p)$ is a closed quantitative literal such that $[\![\theta]\!](T) \bowtie p$ holds, or (c) $\frac{K}{\chi}$ is a valid substitution instance of a rule from R.*

$\frac{H}{\psi}$ is provable from P, notation $P \vdash \frac{H}{\psi}$, if there exists a proof of $\frac{H}{\psi}$ from P.

Before, we said that a 3-valued stable model $\langle \mathsf{CT}, \mathsf{PT} \rangle$ for a PTSS P has to be *justifiably compatible* with the proof system defined by P. By "compatible" we mean that $\langle \mathsf{CT}, \mathsf{PT} \rangle$ has to be consistent with every provable rule. With "justifiable" we require that for each transition in CT and PT there is actually a proof that justifies it. More precisely, we require that (a) for every certain transition in CT there is a proof in P such that all negative hypotheses of the proof are known to hold (i.e. there is no possible transition in PT denying a negative hypothesis), and (b) for every possible transition in PT there is a proof in P such that all negative hypotheses possibly hold (i.e. there is no certain transition in CT denying a negative hypothesis). This is formally stated in the next definition.

Definition 5 (3-valued stable model)**.** *Let $P = (\Sigma, A, R)$ be a PTSS. A tuple $\langle \mathsf{CT}, \mathsf{PT} \rangle$ with $\mathsf{CT} \subseteq \mathsf{PT} \subseteq T(\Sigma_s) \times A \times T(\Sigma_d)$ is a* 3-valued stable model *for P if for every closed positive literal ψ,*
(a) $\psi \in \mathsf{CT}$ iff there is a set N of closed negative literals such that $P \vdash \frac{N}{\psi}$ and $\mathsf{PT} \models N$
(b) $\psi \in \mathsf{PT}$ iff there is a set N of closed negative literals such that $P \vdash \frac{N}{\psi}$ and $\mathsf{CT} \models N$.

The least 3-valued stable model of a PTSS can be constructed using induction [8, 11, 12].

Lemma 1. *Let P be a PTSS. For each ordinal α, define the pair $\langle \mathsf{CT}_\alpha, \mathsf{PT}_\alpha \rangle$ as follows:*

- $\mathsf{CT}_0 = \emptyset$ *and* $\mathsf{PT}_0 = T(\Sigma_s) \times A \times T(\Sigma_d)$.
- *For every non-limit ordinal $\alpha > 0$, define:*

$$\mathsf{CT}_\alpha = \left\{ t \xrightarrow{a} \theta \mid \text{for some set } N \text{ of negative literals, } P \vdash \frac{N}{t \xrightarrow{a} \theta} \text{ and } \mathsf{PT}_{\alpha-1} \models N \right\}$$
$$\mathsf{PT}_\alpha = \left\{ t \xrightarrow{a} \theta \mid \text{for some set } N \text{ of negative literals, } P \vdash \frac{N}{t \xrightarrow{a} \theta} \text{ and } \mathsf{CT}_{\alpha-1} \models N \right\}$$

- *For every limit ordinal α, define $\mathsf{CT}_\alpha = \bigcup_{\beta<\alpha} \mathsf{CT}_\beta$ and $\mathsf{PT}_\alpha = \bigcap_{\beta<\alpha} \mathsf{PT}_\beta$.*

Then: 1. if $\beta \leq \alpha$, $\mathsf{CT}_\beta \subseteq \mathsf{CT}_\alpha$ and $\mathsf{PT}_\beta \supseteq \mathsf{PT}_\alpha$, and 2. there is an ordinal λ such that $\mathsf{CT}_\lambda = \mathsf{CT}_{\lambda+1}$ and $\mathsf{PT}_\lambda = \mathsf{PT}_{\lambda+1}$. Moreover, $\langle \mathsf{CT}_\lambda, \mathsf{PT}_\lambda \rangle$ is the least 3-valued stable model for P.

PTSSs with least 3-valued stable model that are also a 2-valued model are particularly interesting, since this model is actually the only 3-valued stable model [7, 13]. A PTSS P is said to be *complete* if its least 3-valued stable model $\langle \mathsf{CT}, \mathsf{PT} \rangle$ satisfies that $\mathsf{CT} = \mathsf{PT}$ (i.e., the model is also 2-valued). We associate a probabilistic transition system to each complete PTSS.

Definition 6. *Let P be a complete PTSS and let $\langle \mathsf{Tr}, \mathsf{Tr} \rangle$ be its unique 3-valued stable model. We say that Tr is the* transition relation associated to *P. We also define the* PTS associated to *P as the unique PTS $(T(\Sigma_s), A, \rightarrow)$ such that $t \xrightarrow{a} \pi$ if and only if $t \xrightarrow{a} \theta \in \mathsf{Tr}$ and $[\![\theta]\!] = \pi$ for some $\theta \in T(\Sigma_d)$.*

The different examples that we give in the rest of the papers are in terms of a basic probabilistic process algebra. We introduce it here, but address the reader to [8] for an example of a PTSS with richer operators. Signature Σ_s contains the constant 0, representing the stop process, for each action $a \in A$, a unary probabilistic prefix operators $a.\_$ with arity $\mathsf{ar}(a) = d \to s$, and a binary operator $+$, the alternative composition or sum, with arity $\mathsf{ar}(+) = ss \to s$, while Σ_d contains the respective lifted signature, δ, and all binary operators $\bigoplus_{i\in\{1,2\}}[p_i]\theta_i$ which we denote by $\oplus_p$. The semantics is defined with the usual rules:

$$\frac{}{a.\mu \xrightarrow{a} \mu} \qquad \frac{x \xrightarrow{a} \mu}{x+y \xrightarrow{a} \mu} \qquad \frac{y \xrightarrow{a} \mu}{x+y \xrightarrow{a} \mu}$$

4 Bisimulation relations

This work revolves around four different types of bisimulation relations: (i) the usual *(strong) bisimulation* [19] relation on probabilistic system, in which each probabilistic transition should be matched with a single probabilistic transition so that the distributions of both transitions agree on the probabilities of jumping into equivalent states; (ii) the *convex bisimulation* [25] relation, in which the matching is performed instead with a convex combination of transition relations; (iii) the *probability abstracted bisimulation*, in which the matching is performed by a single transition so that the distributions of both transitions agree on jumping to the same equivalent classes of states but not necessarily with the same probability value; and (iv) the *probabilistic obliterated bisimulation*, which represents the usual bisimulation [22] once the probabilistic transition system is abstracted into a traditional labeled transition system in the usual way.

To our knowledge, the probability abstracted bisimulation originates here. Its intention is to strictly preserve the probabilistic structure of a system without caring about the probability values. Thus, probability abstracted bisimulation is consistent with any bisimulation preserving quantitative properties that only tests for positive quantifications, rather than a particular value. Instead, this kind of properties are not preserved by the probabilistic obliterated bisimulation as it is shown below in this section.

In the following we introduce all these relations and discuss their relationship as well as alternative definitions. For the rest of the section we assume given a PTS $P = (\mathsf{T}(\Sigma_s), A, \to)$.

Given a relation $\mathsf{R} \subseteq \mathsf{T}(\Sigma_s) \times \mathsf{T}(\Sigma_s)$, a set $Q \subseteq \mathsf{T}(\Sigma_s)$ is *R-closed* if for all $t \in Q$ and $t' \in \mathsf{T}(\Sigma_s)$, $t\ \mathsf{R}\ t'$ implies $t' \in Q$ (i.e. $\mathsf{R}(Q) \subseteq Q$). It is easy to verify that if two relations $\mathsf{R}, \mathsf{R}' \subseteq \mathsf{T}(\Sigma_s) \times \mathsf{T}(\Sigma_s)$ are such that $\mathsf{R}' \subseteq \mathsf{R}$, then if $Q \subseteq \mathsf{T}(\Sigma_s)$ is R-closed, it is also R'-closed.

Definition 7. *A relation $R \subseteq T(\Sigma_s) \times T(\Sigma_s)$ is a* bisimulation *if it is symmetric and for all $t, t' \in T(\Sigma_s)$, $a \in A$, and $\pi \in \Delta(T(\Sigma_s))$, $t\ R\ t'$ and $t \xrightarrow{a} \pi$ imply that there exists $\pi' \in \Delta(T(\Sigma_s))$ s.t. $t' \xrightarrow{a} \pi'$ and $\pi\ R\ \pi'$, where $\pi\ R\ \pi'$ if and only if for all R-closed $Q \subseteq T(\Sigma_s)$, $\pi(Q) = \pi'(Q)$. The relation $\sim$, called* bisimilarity *or* bisimulation equivalence*, is defined as the smallest relation that includes all bisimulations.*

A *combined transition* $t \xrightarrow{a}_c \pi$ is defined whenever there is a family $\{\pi_i\}_{i\in I} \subseteq \Delta(\mathsf{T}(\Sigma_s))$ and a family $\{p_i\}_{i\in I} \subseteq [0,1]$ such that $t \xrightarrow{a} \pi_i$ for all $i \in I$, $\sum_{i\in I} p_i = 1$ and $\pi = \sum_{i\in I} p_i \pi_i$.

Definition 8. *A relation $R \subseteq T(\Sigma_s) \times T(\Sigma_s)$ is a* convex bisimulation *if it is symmetric and for all $t, t' \in T(\Sigma_s)$, $a \in A$, and $\pi \in \Delta(T(\Sigma_s))$, $t\ R\ t'$ and $t \xrightarrow{a} \pi$ imply that there exists $\pi' \in \Delta(T(\Sigma_s))$ s.t. $t' \xrightarrow{a}_c \pi'$ and $\pi\ R\ \pi'$. The relation $\sim_c$, called* convex bisimilarity *or* convex bisimulation equivalence*, is defined as the smallest relation that includes all convex bisimulations.*

Definition 9. *A relation $R \subseteq T(\Sigma_s) \times T(\Sigma_s)$ is a* probability abstracted bisimulation *if it is symmetric and for all $t, t' \in T(\Sigma_s)$, $a \in A$, and $\pi \in \Delta(T(\Sigma_s))$, $t\ R\ t'$ and $t \xrightarrow{a} \pi$ imply that there exists $\pi' \in \Delta(T(\Sigma_s))$ s.t. $t' \xrightarrow{a} \pi'$*

and for all R-closed $Q \subseteq T(\Sigma_s)$, $\pi(Q) > 0$ iff $\pi'(Q) > 0$. The relation $\sim_a$, called probability abstracted bisimilarity *or* probability abstracted bisimulation equivalence*, is defined as the smallest relation that includes all probability abstracted bisimulations.*

Notice that the transfer property in this last case follows the same structure as the bisimulation, only that it only requires that $\pi(Q) > 0$ iff $\pi'(Q) > 0$ for all R-closed, instead of $\pi(Q) = \pi'(Q)$.

Definition 10. *A relation $R \subseteq T(\Sigma_s) \times T(\Sigma_s)$ is a* probability obliterated bisimulation *if it is symmetric and for all $t, t' \in T(\Sigma_s)$, $a \in A$, and $\pi \in \Delta(T(\Sigma_s))$, $t \; R \; t'$ and $t \xrightarrow{a} \pi$ imply that for all R-closed $Q \subseteq T(\Sigma_s)$ with $\pi(Q) > 0$, there exists $\pi' \in \Delta(T(\Sigma_s))$ s.t. $t' \xrightarrow{a} \pi'$ and $\pi'(Q) > 0$. The relation $\sim_o$, called* probability obliterated bisimilarity *or* probability obliterated bisimulation equivalence*, is defined as the smallest relation that includes all probability obliterated bisimulations.*

Compare this last definition with Def. 9. While for probability abstracted bisimulation we require that there is a single matching transition $t' \xrightarrow{a} \pi'$ so that π' gives some positive probability to all R-closed sets exactly whenever π does, the definition of probability obliterated bisimulation permits to choose different matching transitions for each R-closed set that measures positively on π.

It is well known that $\sim$ and $\sim_c$ are equivalences relations and that they also are, respectively, a bisimulation relation and a convex bisimulation relation. The fact that $\sim_o$ is also an equivalence relation and itself a probability obliterated bisimulation follows from Lemma 4 which state that it agrees with Park & Milner's bisimulation. The same properties can be proven for probability abstracted bisimulation:

Lemma 2. *$\sim_a$ is an equivalence relation and is itself a probability abstracted bisimulation.*

Similarly to the bisimulation [3, Prop 3.4.4], the probability abstracted bisimulation has a characterization in terms of an *abstract* weight function. This alternative characterization is the one used in the proof of Theorem 4 and that is why we present it in this paper.

Given a relation $\mathsf{R} \subseteq \mathsf{T}(\Sigma_s) \times \mathsf{T}(\Sigma_s)$, we define $\equiv^w_{\mathsf{R}} \in \Delta(\mathsf{T}(\Sigma_s)) \times \Delta(\mathsf{T}(\Sigma_s))$ as follows. For all $\pi, \pi' \in \Delta(\mathsf{T}(\Sigma_s))$, $\pi \equiv^w_{\mathsf{R}} \pi'$ if there is an *abstract weight function* $w : (\mathsf{T}(\Sigma_s) \times \mathsf{T}(\Sigma_s)) \to [0,1]$ s.t. for all $t, t' \in \mathsf{T}(\Sigma_s)$, (i) $w(t, \mathsf{T}(\Sigma_s)) > 0$ iff $\pi(t) > 0$, (ii) $w(\mathsf{T}(\Sigma_s), t') > 0$ iff $\pi'(t') > 0$, and (iii) $w(t, t') > 0$ implies $t \; \mathsf{R} \; t'$.

Lemma 3. *For all $t, t' \in T(\Sigma_s)$, $t \sim_a t'$ if and only if there is a symmetric relation $R \subseteq T(\Sigma_s) \times T(\Sigma_s)$ with $t \; R \; t'$ such that for all $t_1, t_2 \in T(\Sigma_s)$, $a \in A$, and $\pi_1 \in \Delta(T(\Sigma_s))$, $t_1 \; R \; t_2$ and $t_1 \xrightarrow{a} \pi_1$ imply that there exists $\pi_2 \in \Delta(T(\Sigma_s))$ s.t. $t_2 \xrightarrow{a} \pi_2$ and $\pi_1 \equiv^w_R \pi_2$.*

The next lemma shows that the probability obliterated bisimulation agrees with Park & Milner's bisimulation. Denote $t \overset{a}{\rightsquigarrow} t'$ iff there is π such that $t \xrightarrow{a} \pi$ and $\pi(t') > 0$. Notice that this notation precisely defines the usual abstraction of probabilistic transition systems into labeled transition systems in which all information regarding the probability distribution is lost except from the fact that one state can reach another state with positive probability after a transition.

Lemma 4. *For all $t, t' \in T(\Sigma_s)$, $t \sim_o t'$ iff there is a symmetric relation $R \subseteq T(\Sigma_s) \times T(\Sigma_s)$ with $t \; R \; t'$ s.t. for all $t_1, t_2, t_1' \in T(\Sigma_s)$ and $a \in A$, $t_1 \; R \; t_2$ and $t_1 \overset{a}{\rightsquigarrow} t_1'$ imply that there exists $t_2' \in T(\Sigma_s)$ s.t. $t_2 \overset{a}{\rightsquigarrow} t_2'$ and $t_1' \; R \; t_2'$.*

Finally we state the relation among the different bisimulations

Lemma 5. *The following inclusions hold and are proper: $\sim \subsetneq \sim_c \subsetneq \sim_o$ and $\sim \subsetneq \sim_a \subsetneq \sim_o$. Besides $\sim_c$ and $\sim_a$ are incomparable.*

In fact the results can be proved stronger as we explain in the following. Any bisimulation relation is also a convex bisimulation, which follow from the fact that $t \xrightarrow{a} \pi$ implies $t \xrightarrow{a}_c \pi$. Any convex bisimulation is also a probability obliterated bisimulation since $t \xrightarrow{a}_c \pi$ with $\pi(Q) > 0$ implies that there is a

π' such that $t \xrightarrow{a} \pi'$ and $\pi'(Q) > 0$. Any bisimulation is also a probability abstracted bisimulation since $\pi \mathsf{R} \pi'$ implies $\pi(Q) > 0$ iff $\pi'(Q) > 0$ for all R-closed Q. Finally, any probability abstracted bisimulation is also a probability obliterated bisimulation since, for a given π and R, the existence of a π' s.t. $t' \xrightarrow{a} \pi'$ and $\pi(Q) > 0$ iff $\pi'(Q) > 0$ for all R-closed Q, guarantees that, for all R-closed Q with $\pi(Q) > 0$ there is a π' s.t. $t' \xrightarrow{a} \pi'$ and $\pi'(Q) > 0$.

Notice that $\mathtt{t}_1 = a.(\boldsymbol{b.0}) + a.(\boldsymbol{c.0})$ and $\mathtt{t}_2 = \mathtt{t}_1 + a.(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0})$ are convex bisimilar but not probability abstracted bisimilar. Besides, notice that $\mathtt{t}_3 = a.(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0})$ and $\mathtt{t}_4 = a.(\boldsymbol{b.0} \oplus_{0.1} \boldsymbol{c.0})$ are probability abstracted bisimilar but not convex bisimilar. These examples not only show that $\sim_c$ and $\sim_a$ are incomparable, but also that all stated inclusions are proper.

In the rest of the section we present logical characterizations for the different bisimulation equivalences. This work has already been done for bisimulation [10, 16] and convex bisimulation [16]. We adopt here the two-level logic style of [10].

We define the logic $\mathcal{L}_b$ as the set of all formulas with the following syntax:

$$\phi := \top \mid \langle a \rangle \psi \mid \langle a \rangle_c \psi \mid \bigwedge_{i \in I} \phi_i \mid \neg \phi \qquad\qquad \psi := [\phi]_p \mid \bigsqcap_{i \in I} \psi_i$$

where $a \in A$, $p \in [0,1] \cap \mathbb{Q}$, and I is any index set. The logic $\mathcal{L}_c$ contains all formulas of $\mathcal{L}_b$ without the modality $\langle a \rangle_-$. The logic $\mathcal{L}_a$ contains all formulas of $\mathcal{L}_b$ without modalities $\langle a \rangle_{c-}$ and $[\_]_p$ for all $p > 0$ (i.e. it only accepts $[\_]_0$ among this type of modalities.) Finally, the logic $\mathcal{L}_o$ contains all formulas of $\mathcal{L}_a$ without $\bigsqcap_{i \in I -}$.

The semantics of $\mathcal{L}_b$ is defined with the satisfaction relation $\models$ on a PTS $P = (\mathsf{T}(\Sigma_s), A, \rightarrow)$ as follows.

(i) $t \models \top$ for all $t \in \mathsf{T}(\Sigma_s)$

(ii) $t \models \langle a \rangle \psi$ if there is $t \xrightarrow{a} \pi$ s.t. $\pi \models \psi$

(iii) $t \models \langle a \rangle_c \psi$ if there is $t \xrightarrow{a}_c \pi$ s.t. $\pi \models \psi$

(iv) $t \models \bigwedge_{i \in I} \phi_i$ if $t \models \phi_i$ for all $i \in I$

(v) $t \models \neg \phi$ if $t \not\models \phi$

(vi) $\pi \models [\phi]_p$ if $\pi(\{t \in \mathsf{T}(\Sigma_s) \mid t \models \phi\}) > p$

(vii) $\pi \models \bigsqcap_{i \in I} \psi_i$ if $\pi \models \psi_i$ for all $i \in I$

The semantics of the other logics is defined in the same way but restricted to the respective operators.

For $\chi \in \{b, c, a, o\}$, let $\mathcal{L}_\chi(t) = \{\phi \in \mathcal{L}_\chi \mid t \models \phi\}$, for all $t \in \mathsf{T}(\Sigma_s)$, and $\mathcal{L}_\chi(\pi) = \{\psi \in \mathcal{L}_\chi \mid \pi \models \psi\}$, for all $\pi \in \Delta(\mathsf{T}(\Sigma_s))$. We write $t_1 \sim_{\mathcal{L}_\chi} t_2$ iff $\mathcal{L}_\chi(t_1) = \mathcal{L}_\chi(t_2)$ and $\pi_1 \sim_{\mathcal{L}_\chi} \pi_2$ iff $\mathcal{L}_\chi(\pi_1) = \mathcal{L}_\chi(\pi_2)$. Then, we have the following characterization theorem.

Theorem 1. *For all $\chi \in \{b, c, a, o\}$ and for all $t_1, t_2 \in \mathsf{T}(\Sigma_s)$, $t_1 \sim_\chi t_2$ iff $t_1 \sim_{\mathcal{L}_\chi} t_2$ (where $\sim_b = \sim$).*

Let $\mathtt{t}_1$, $\mathtt{t}_2$, $\mathtt{t}_3$, and $\mathtt{t}_4$ be as before. Recall $\mathtt{t}_1 \sim_c \mathtt{t}_2$ and $\mathtt{t}_3 \sim_a \mathtt{t}_4$. Notice that $\langle a \rangle([\langle b \rangle \top]_{0.5} \sqcap [\langle c \rangle \top]_{0.5})$ distinguish $\mathtt{t}_1$ from $\mathtt{t}_2$, while $\langle a \rangle_c([\langle b \rangle \top]_{0.5} \sqcap [\langle c \rangle \top]_{0.5})$ is satisfied by both $\mathtt{t}_1$ and $\mathtt{t}_2$. That is why $\langle a \rangle_-$ is not an operator of $\mathcal{L}_c$. Notice $[\langle b \rangle \top]_{0.5}$ distinguishes the distribution $[\![b.\mathbf{0} \oplus_{0.5} c.\mathbf{0}]\!]$ from $[\![b.\mathbf{0} \oplus_{0.1} c.\mathbf{0}]\!]$, while $[\langle b \rangle \top]_0$ does not (but it does distinguish them from e.g. $[\![c.\mathbf{0}]\!]$). Thus $\langle a \rangle [\langle b \rangle \top]_{0.5}$ distinguishes $\mathtt{t}_3$ from $\mathtt{t}_4$. That is why $[\_]_p$ is not an operator of $\mathcal{L}_a$ if $p > 0$. Finally, notice that $\langle a \rangle([\langle b \rangle \top]_0 \sqcap [\langle c \rangle \top]_0)$ distinguishes $\mathtt{t}_5 = a.(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0})$ from $\mathtt{t}_6 = a.\boldsymbol{b.0} + a.\boldsymbol{c.0}$, and observe that $\mathtt{t}_5 \sim_o \mathtt{t}_6$. However, neither $\langle a \rangle [\langle b \rangle \top]_0$ nor $\langle a \rangle [\langle c \rangle \top]_0$ can distinguish them. That is why $\bigsqcap_{i \in I -}$ is not an operator of $\mathcal{L}_o$.

5 Formats

In this section we introduce rule and specification formats that guarantee that each bisimulation equivalences discussed in the previous section is a congruence for every operator whose semantics is defined

within the respective rule of the specification format. In particular, the format *ntμfθ/ntμxθ*, which ensures that bisimulation equivalence is a congruence for all operator in such format, has been already introduced in [9] and finally revised in [8]. We present here its more general form.

The following definition is important to ensure a symmetric treatment of variables and terms within the format. Let $\{Y_l\}_{l\in L}$ be a family of sets of state term variables with the same cardinality. The l-th element of a tuple $\vec{y}$ is denoted by $\vec{y}(l)$. For a set of tuples $T = \{\vec{y_i} \mid i \in I\}$ we denote the l-th projection by $\Pi_l(T) = \{\vec{y_i}(l) \mid i \in I\}$. Fix a set $\mathsf{Diag}\{Y_l\}_{l\in L} \subseteq \prod_{l\in L} Y_l$ such that: (i) for all $l \in L$, $\Pi_l(\mathsf{Diag}\{Y_l\}_{l\in L}) = Y_l$; and (ii) for all $\vec{y},\vec{y'} \in \mathsf{Diag}\{Y_l\}_{l\in L}$, $(\exists l \in L : \vec{y}(l) = \vec{y'}(l)) \Rightarrow \vec{y} = \vec{y'}$. Property (ii) ensures that different tuples $\vec{y},\vec{y'} \in \mathsf{Diag}\{Y_l\}_{l\in L}$ differ in all positions, and by property (i) every variable of every Y_l is used in (exactly) one $\vec{y} \in \mathsf{Diag}\{Y_l\}_{l\in L}$. Diag stands for "diagonal", following the intuition that each $\vec{y}$ represents a coordinate in the space $\prod_{l\in L} Y_l$, so that $\mathsf{Diag}\{Y_l\}_{l\in L}$ can be seen as the line that traverses the main diagonal of the space. Therefore, notice that, for $Y_l = \{y_l^0, y_l^1, y_l^2, \ldots\}$, a possible definition for the set $\mathsf{Diag}\{Y_l\}_{l\in L}$ is $\{(y_1^0, y_2^0, \ldots, y_L^0), (y_1^1, y_2^1, \ldots, y_L^1), (y_1^2, y_2^2, \ldots, y_L^2), \ldots\}$. In addition, we use the following notation: $t(\zeta_1, \ldots, \zeta_n)$ denotes a term that only has variables in the set $\{\zeta_1, \ldots, \zeta_n\}$, that is $\mathcal{V}(t(\zeta_1, \ldots, \zeta_n)) \subseteq \{\zeta_1, \ldots, \zeta_n\}$, and moreover, $t(\zeta'_1, \ldots, \zeta'_n)$ denotes the same term as $t(\zeta_1, \ldots, \zeta_n)$ in which each variable ζ_i has been replaced by ζ'_i.

Definition 11. *Let $P = (\Sigma, A, R)$ be a PTSS. A rule $r \in R$ is in ntμfθ* format *if it has the following form*

$$\frac{\bigcup_{m\in M}\{t_m(\vec{z}) \xrightarrow{a_m} \mu_m^{\vec{z}} \mid \vec{z} \in \mathcal{Z}\} \cup \bigcup_{n\in N}\{t_n(\vec{z}) \xrightarrow{b_n} \not\rightarrow \mid \vec{z} \in \mathcal{Z}\} \cup \{\theta_l(Y_l) \trianglerighteq_{l,k} p_{l,k} \mid l \in L, k \in K_l\}}{f(\zeta_1, \ldots, \zeta_{\mathsf{rk}(f)}) \xrightarrow{a} \theta}$$

with $\trianglerighteq_{l,k} \in \{>, \geq\}$ for all $l \in L$ and $k \in K_l$, and $\mathcal{Z} = \mathsf{Diag}\{Y_l\}_{l\in L} \times \prod_{\zeta\in W}\{\zeta\}$, with $W \subseteq \mathcal{V} \cup \mathcal{V}_d \setminus \bigcup_{l\in L} Y_l$. In addition, it has to satisfy the following conditions:

1. *Each set Y_l should be at least countably infinite, for all $l \in L$, and the cardinality of L should be strictly smaller than that of the Y_l's.*
2. *All variables $\zeta_1, \ldots, \zeta_{\mathsf{rk}(f)}$ are different.*
3. *All variables $\mu_m^{\vec{z}}$, with $m \in M$ and $\vec{z} \in \mathcal{Z}$, are different and $\{\zeta_1, \ldots, \zeta_{\mathsf{rk}(f)}\} \cap \{\mu_m^{\vec{z}} \mid \vec{z} \in \mathcal{Z}, m \in M\} = \emptyset$.*
4. *For all $l \in L$, $Y_l \cap \{\zeta_1, \ldots, \zeta_{\mathsf{rk}(f)}\} = \emptyset$, and $Y_l \cap Y_{l'} = \emptyset$ for all $l' \in L$, $l \neq l'$.*
5. *For all $m \in M$, the set $\{\mu_m^{\vec{z}} \mid \vec{z} \in \mathcal{Z}\} \cap (\mathcal{V}(\theta) \cup (\bigcup_{l\in L} \mathcal{V}(\theta_l)) \cup W)$ is finite.*
6. *For all $l \in L$, the set $Y_l \cap (\mathcal{V}(\theta) \cup \bigcup_{l'\in L} \mathcal{V}(\theta_{l'}))$ is finite.*

A rule $r \in R$ is in ntμxθ format *if its form is like above but has a conclusion of the form $x \xrightarrow{a} \theta$ and, in addition, it satisfies the same conditions as above, except that whenever we write $\{\zeta_1, \ldots, \zeta_{\mathsf{rk}(f)}\}$, we should write $\{x\}$. P is in ntμfθ* format *if all its rules are in ntμfθ format. P is in ntμfθ/ntμxθ* format *if all its rules are in either ntμfθ format or ntμxθ format.*

The rationale behind each of the restrictions are discussed in [8] in depth (see also [9]). In the following we briefly summarize it. Variables $\zeta_1, \ldots, \zeta_{\mathsf{rk}(f)}$ in the source of the conclusion, all variables $\mu_m^{\vec{z}}$ in the target of the positive premises, and all variables in the sets Y_l, $l \in L$, as part of the measurable sets in the quantitative premises, are binding. That is why all of them are requested to be different, which is stated in conditions 2, 3, and 4. If Y_l is finite, quantitative premises will allow to count the minimum number of terms that gather certain probabilities. This goes against the spirit of bisimulation that measures equivalence classes of terms regardless of the size of them. Therefore Y_l needs to be infinite (condition 1). Condition 5 ensures that, for each $m \in M$ there are sufficiently many distribution variables in the set $\{\mu_m^{\vec{z}} \mid \vec{z} \in \mathcal{Z}\}$ to be freely instantiated. The use of a distribution variable in a quantitative premise may disclose part of the structural nature of the distribution term that substitutes such variable. Thus, for instance, if all variables $\mu_m^{\vec{z}}$ are used in different quantitative premises together with some lookahead, we

may restrict the syntactic form of the eventually substituted distribution terms, hence revealing unwanted differences. A similar situation arises with the use of variables in Y_l for all $l \in L$, hence condition 6. The precise understanding of conditions 5 and 6 requires a rather lengthy explanation that is beyond the scope of this paper. The reader is referred to [8, 9] for details.

All congruence theorems in this article apply only to PTSSs whose rules are *well-founded*. A rule r is *well-founded* if there is no infinite backward chain in the *dependency graph* $G_r = (V, E)$ of r defined by $V = \mathcal{V} \cup \mathcal{V}_d$ and $E = \{\langle \zeta, \mu \rangle \mid (t \xrightarrow{a} \mu) \in \text{pprem}(r), \zeta \in \mathcal{V}(t)\} \cup \{\langle \zeta, y \rangle \mid (\theta(Y) \trianglerighteq p) \in \text{qprem}(r), \zeta \in \mathcal{V}(\theta), y \in Y\}$. A PTSS is called *well-founded* if all its rules are well-founded.

The full proof of the following theorem can be found in [8].

Theorem 2. *Let $P = (\Sigma, A, R)$ be a complete well-founded PTSS in nt$\mu f\theta$/nt$\mu x\theta$ format. Then, the bisimulation equivalence is a congruence for all operators defined in P.*

The *nt$\mu f\theta$/nt$\mu x\theta$* format is still too general to preserve the other (weaker) bisimulation equivalences presented in Sec. 4. In the reminder of the section, we will discuss through appropriate examples how the *nt$\mu f\theta$/nt$\mu x\theta$* format should be further restricted or modified so that the other bisimulation equivalences are congruences for the resulting restricted formats.

We focus first on convex bisimulation. For this consider the terms $\mathtt{t}_1 = a.(\boldsymbol{b}.\mathbf{0}) + a.(\boldsymbol{c}.\mathbf{0})$ and $\mathtt{t}_2 = \mathtt{t}_1 + a.(\boldsymbol{b}.\mathbf{0} \oplus_{0.5} \boldsymbol{c}.\mathbf{0})$. Notice that $\mathtt{t}_1 \sim_c \mathtt{t}_2$. Consider a possible extension of our running example with a unary operator f with the following *nt$\mu f\theta$* rule:

$$\frac{x \xrightarrow{a} \mu \quad \mu(Y) \geq 0.5 \quad \{y \xrightarrow{b} \mu_y \mid y \in Y\} \quad \mu(Y') \geq 0.5 \quad \{y' \xrightarrow{c} \mu_{y'} \mid y' \in Y'\}}{f(x) \xrightarrow{a} \mathbf{0}} \tag{1}$$

Since $\mathtt{t}_2 \xrightarrow{a} (\boldsymbol{b}.\mathbf{0} \oplus_{0.5} \boldsymbol{c}.\mathbf{0})$, $f(\mathtt{t}_2) \xrightarrow{a} \mathbf{0}$. However it is easy to see that $f(\mathtt{t}_1)$ cannot perform any transition. Therefore $f(\mathtt{t}_1) \not\sim_c f(\mathtt{t}_2)$.

The problem arises precisely because, in order to show that $\mathtt{t}_1 \sim_c \mathtt{t}_2$, transition $\mathtt{t}_2 \xrightarrow{a} (\boldsymbol{b}.\mathbf{0} \oplus_{0.5} \boldsymbol{c}.\mathbf{0})$ is matched with the appropriate convex combination of the transitions $\mathtt{t}_1 \xrightarrow{a} \boldsymbol{b}.\mathbf{0}$ and $\mathtt{t}_1 \xrightarrow{a} \boldsymbol{c}.\mathbf{0}$. Thus, we need that a quantitative premise guarantees that the test is produced on a convex combination of target distributions rather than on a single target distribution. An appropriate modification of such rule would be to replace it by a family of rules of the form

$$\frac{\{x \xrightarrow{a} \mu_n \mid n \in \mathbb{N}\} \quad \left(\bigoplus_{n\in\mathbb{N}} [p_n]\mu_n\right)(Y) \geq 0.5 \quad \{y \xrightarrow{b} \mu_y \mid y \in Y\} \quad \left(\bigoplus_{n\in\mathbb{N}} [p_n]\mu_n\right)(Y') \geq 0.5 \quad \{y' \xrightarrow{c} \mu_{y'} \mid y' \in Y'\}}{f(x) \xrightarrow{a} \mathbf{0}},$$

one for each $\{p_n\}_{n\in\mathbb{N}}$ such that $\sum_{n\in\mathbb{N}} p_n = 1$ and each $p_i \in [0, 1] \cap \mathbb{Q}$.

Consider now that the semantic of f is defined by the rule

$$\frac{x \xrightarrow{a} \mu}{f(x) \xrightarrow{a} \boldsymbol{a}.\mu} \tag{2}$$

and notice that $f(\mathtt{t}_2) \xrightarrow{a} \boldsymbol{a}.(\boldsymbol{b}.\mathbf{0} \oplus_{0.5} \boldsymbol{c}.\mathbf{0})$. However, the only two possible transitions for $f(\mathtt{t}_1)$ are $f(\mathtt{t}_1) \xrightarrow{a} \boldsymbol{a}.\boldsymbol{b}.\mathbf{0}$ and $f(\mathtt{t}_1) \xrightarrow{a} \boldsymbol{a}.\boldsymbol{c}.\mathbf{0}$, and there is no $p \in [0, 1]$ such that $\boldsymbol{a}.\boldsymbol{b}.\mathbf{0} \oplus_p \boldsymbol{a}.\boldsymbol{c}.\mathbf{0} \sim_c \boldsymbol{a}.(\boldsymbol{b}.\mathbf{0} \oplus_{0.5} \boldsymbol{c}.\mathbf{0})$. For this reason, we will require that a target of a positive premise does not appear in a d-sorted position of a subterm in the target of the conclusion.

For the next example, we consider an additional unary d-sorted operator g and the following rules

$$\frac{x \xrightarrow{a} \mu \qquad g(\mu) \xrightarrow{b} \mu'}{f(x) \xrightarrow{a} \mathbf{0}} \qquad \frac{\mu(Y) > 0 \quad \{y \xrightarrow{b} \mu \mid y \in Y\} \quad \mu(Y') > 0 \quad \{y' \xrightarrow{c} \mu' \mid y' \in Y'\}}{g(\mu) \xrightarrow{b} \mathbf{0}} \tag{3}$$

Notice that $g(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0}) \xrightarrow{b} \mathbf{0}$. Therefore $f(\mathtt{t}_2) \xrightarrow{a} \mathbf{0}$. However, neither $g(\boldsymbol{b.0})$ nor $g(\boldsymbol{c.0})$ can perform any transition, and as a consequence $f(\mathtt{t}_1)$ cannot perform any transition either. Hence, $f(\mathtt{t}_1) \not\sim_c f(\mathtt{t}_2)$. For this reason we will require that a target of a positive premise does not appear in the source of a positive or negative premise.

Suppose now that g is a binary s-sorted operator and consider the following rules

$$\frac{x \xrightarrow{a} \mu}{f(x) \xrightarrow{a} \boldsymbol{g}(\mu,\mu)} \qquad \frac{x_1 \xrightarrow{b} \mu_1 \qquad x_2 \xrightarrow{c} \mu_2}{g(x_1,x_2) \xrightarrow{a} \mathbf{0}} \tag{4}$$

Notice that the only possible transitions for $f(\mathtt{t}_1)$ are $f(\mathtt{t}_1) \xrightarrow{a} \boldsymbol{g}(\boldsymbol{b.0},\boldsymbol{b.0})$ and $f(\mathtt{t}_1) \xrightarrow{a} \boldsymbol{g}(\boldsymbol{c.0},\boldsymbol{c.0})$. Moreover, notice that $\boldsymbol{g}(\boldsymbol{b.0},\boldsymbol{b.0}) \sim_c \boldsymbol{g}(\boldsymbol{c.0},\boldsymbol{c.0}) \sim_c \mathbf{0}$. However, $f(\mathtt{t}_2) \xrightarrow{a} \boldsymbol{g}(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0}, \boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0})$, and it is not difficult to see that $\boldsymbol{g}(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0}, \boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0}) \sim_c (\boldsymbol{a.0} \oplus_{0.25} \mathbf{0})$. Therefore, $f(\mathtt{t}_1) \not\sim_c f(\mathtt{t}_2)$. In this case, the problem seems to arise because the same distribution variable occurs in the target of the conclusion of the first rule in two different s-sorts positions of the target distribution term. However, the problem is not so general. Notice that if the target in the conclusion is replaced by the term $\boldsymbol{g}(\mu,\boldsymbol{c.0}) \oplus_p \boldsymbol{g}(\boldsymbol{b.0},\mu)$ we would have $f(\mathtt{t}_1) \sim_c f(\mathtt{t}_2)$. The difference arises from the fact that in the interpretation of $\boldsymbol{g}(\theta,\theta)$ the probability distribution $[\![\theta]\!]$ multiplies with itself. This is not the case in the interpretation of $\boldsymbol{g}(\theta,\boldsymbol{c.0}) \oplus_p \boldsymbol{g}(\boldsymbol{b.0},\theta)$ where the two instances of $[\![\theta]\!]$ are summed up. Thus, we will actually request that the target of the conclusion is *linear* with respect to each distribution variable on a target of a positive premise.

Definition 12. *A distribution term $\theta \in \mathbb{T}(\Sigma_d)$ is* linear for a set $V \subseteq \mathcal{V}_d$ *if (i) $\theta \in \mathsf{T}(\Sigma_d) \cup \mathcal{V}_d \cup \{\delta(x) \mid x \in \mathcal{V}\}$. (ii) $\theta = \bigoplus_{i\in I}[p_i]\theta_i$ and θ_i is linear for V, for all $i \in I$, (iii) $\theta = f(\theta_1,\ldots,\theta_n)$, for all $i \in I$, θ_i is linear for V, and $\mathcal{V}(\theta_i) \cap \mathcal{V}(\theta_j) \cap V = \emptyset$, for all $i,j \in \{1,\ldots,n\}$ and $i \neq j$,*

Definition 13. *Let $P = (\Sigma, A, R)$ be a PTSS. A rule $r \in R$ is in* convex *ntμfθ* format *if has the form*

$$\frac{\bigcup_{m\in M}\{t_m(\vec{z}) \xrightarrow{a_m} \mu_m^{\vec{z}} \mid \vec{z} \in \mathcal{Z}\} \qquad \bigcup_{n\in N}\{t_n(\vec{z}) \overset{b_n}{\not\longrightarrow} \mid \vec{z} \in \mathcal{Z}\}}{\bigcup_{\tilde{m}\in\tilde{M}}\{t_{\tilde{m}}(\vec{z}_{\tilde{m}}) \xrightarrow{a_{\tilde{m}}} \mu_i^{\tilde{m}} \mid i \in \mathbb{N}\} \qquad \bigcup_{\tilde{m}\in\tilde{M}}\{(\bigoplus_{i\in\mathbb{N}}[p_i^{\tilde{m}}]\mu_i^{\tilde{m}})(Y_l) \trianglerighteq_{l,k} p_{l,k} \mid l \in L_{\tilde{m}}, k \in K_l\}}{f(\zeta_1,\ldots,\zeta_{\mathsf{rk}(f)}) \xrightarrow{a} \theta}$$

with $L = \cup_{\tilde{m}\in\tilde{M}} L_{\tilde{m}}$, $L_{\tilde{m}} \cap L_{\tilde{m}'} = \emptyset$ whenever $\tilde{m} \neq \tilde{m}'$, $\trianglerighteq_{l,k} \in \{>,\geq\}$ for all $l \in L$ and $k \in K_l$, $\mathcal{Z} = \mathsf{Diag}\{Y_l\}_{l\in L} \times \prod_{\zeta\in W}\{\zeta\}$, with $W \subseteq \mathcal{V} \cup \mathcal{V}_d \setminus \bigcup_{l\in L} Y_l$. In addition, it should also satisfy conditions 1 to 6 in Def. 11 and the following extra conditions:

7. *For every $\tilde{m} \in \tilde{M}$, the family $\{p_i^{\tilde{m}}\}_{i\in\mathbb{N}} \subseteq [0,1] \cap \mathbb{Q}$ and $\sum_{i\in\mathbb{N}} p_i^{\tilde{m}} = 1$*
8. *For every $\tilde{m} \in \tilde{M}$, there is exactly one $j \in \mathbb{N}$ such that $\mu_j^{\tilde{m}} = \mu_m^{\vec{z}}$ for some $m \in M$ and $\vec{z} \in \mathcal{Z}$, in which case also $t_{\tilde{m}}(\vec{z}_{\tilde{m}}) \xrightarrow{a_{\tilde{m}}} \mu_j^{\tilde{m}} = t_m(\vec{z}) \xrightarrow{a_m} \mu_m^{\vec{z}}$. Moreover $\{\mu_i^{\tilde{m}} \mid i \in \mathbb{N}\} \cap \{\mu_i^{\tilde{m}'} \mid i \in \mathbb{N}\} = \emptyset$ for all $\tilde{m} \neq \tilde{m}'$, and $\{\mu_i^{\tilde{m}} \mid i \in \mathbb{N}\} \cap \{\zeta_1,\ldots,\zeta_{\mathsf{rk}(f)}\} = \emptyset$.*
9. *No variable $\mu_m^{\vec{z}}$, with $m \in M$ and $\vec{z} \in \mathcal{Z}$, appears in the source of a premise (i.e. in the set W) or in a d-sorted position of a subterm in the target of the conclusion θ.*
10. *θ is linear for $\{\mu_m^{\vec{z}} \mid m \in M, \vec{z} \in \mathcal{Z}\}$.*

A rule $r \in R$ is in convex *ntμxθ* format *if its form is like above but has a conclusion of the form $x \xrightarrow{a} \theta$ and it satisfies the same conditions, except that whenever we write $\{\zeta_1,\ldots,\zeta_{\mathsf{rk}(f)}\}$, we should write $\{x\}$. A set of convex ntμfθ/ntμxθ rules R is* convex closed *if for all $r \in R$, for any term $\bigoplus_{i\in\mathbb{N}}[p_i^{\tilde{m}}]\mu_i^{\tilde{m}}$ appearing in a quantitative premise of r and any family $\{q_i\}_{i\in\mathbb{N}} \subseteq [0,1] \cap \mathbb{Q}$ such that $\sum_{i\in\mathbb{N}} q_i = 1$, then the rule r' obtained by replacing each occurrence of $\bigoplus_{i\in\mathbb{N}}[p_i^{\tilde{m}}]\mu_i^{\tilde{m}}$ in r by $\bigoplus_{i\in\mathbb{N}}[q_i]\mu_i^{\tilde{m}}$ is also in R. A PTSS $P = (\Sigma, A, R)$ is in* convex *ntμfθ/ntμxθ* format *if all rules in R are in convex ntμfθ/ntμxθ format and R is convex closed.*

The problem indicated in rule (1) is attacked with the requirement of having sets $\{t_{\tilde{m}}(\vec{z}_{\tilde{m}}) \xrightarrow{a_{\tilde{m}}} \mu_i^{\tilde{m}} \mid i \in \mathbb{N}\}$ as positive premises with which the convex closures $\bigoplus_{i\in\mathbb{N}}[p_i^{\tilde{m}}]\mu_i^{\tilde{m}}$ can be constructed, plus the request that the set of rules is convex closed. Notice that condition 8 states that these sets of positive premises are only used to construct such distribution terms and are only linked to the "actual" positive premises in $\bigcup_{m\in M}\{t_m(\vec{z}) \xrightarrow{a_m} \mu_m^{\vec{z}} \mid \vec{z} \in \mathcal{Z}\}$ through a single transition $t_{\tilde{m}}(\vec{z}_{\tilde{m}}) \xrightarrow{a_{\tilde{m}}} \mu_j^{\tilde{m}}$.

Rules like (2) and the left rule on (3) are excluded on condition 9 since no variable of a positive premise can be used in the source of a premise (excluding (3)) or in a d-sort position in the target of the conclusion (excluding (2)). Finally, rules like on the left of (4) are excluded by requesting that the target of the conclusion is linear (condition 10).

Now, we can state the congruence theorem for convex bisimulation equivalence.

Theorem 3. *Let P be a complete well-founded PTSS in convex ntμfθ/ntμxθ format. Then, convex bisimulation equivalence is a congruence for all operators defined by P.*

We focus now on the probability abstracted bisimulation. Notice that the terms $\mathfrak{t}_3 = a.(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0})$ and $\mathfrak{t}_4 = a.(\boldsymbol{b.0} \oplus_{0.1} \boldsymbol{c.0})$ are probability abstracted bisimilar, i.e., $\mathfrak{t}_3 \sim_a \mathfrak{t}_4$. Consider now the unary operator f whose semantics is defined with rule (1). It should not be difficult so see that $f(\mathfrak{t}_3) \xrightarrow{a} \mathbf{0}$ while $f(\mathfrak{t}_4)$ cannot perform any transition. Therefore $f(\mathfrak{t}_3) \not\sim_a f(\mathfrak{t}_4)$. The problem is a consequence of the fact that the quantitative premises are tested against non-zero values which may distinguish distributions with the same support set but mapping into different probability values. Thus, in order to preserve probability abstracted bisimulation equivalence, the only extra restriction that we ask to a rule in *ntμfθ/ntμxθ* format is that none of its quantitative premises test against a value different from 0.

Definition 14. *A PTSS $P = \langle \Sigma, A, R\rangle$ is in* probability abstracted *ntμfθ/ntμxθ* format *if it is in ntμfθ/ntμxθ format and for every rule $r \in R$ and quantitative premise $\theta(Y) \trianglerighteq p \in qprem(r)$, $p = 0$.*

The proof of the congruence theorem for probability abstracted bisimulation equivalence (Theorem 4 below) follows closely the lines of the proof of Theorem 2 as given in [8].

Theorem 4. *Let P be a complete well-founded PTSS in probability abstracted ntμfθ/ntμxθ format. Then, the probability abstracted bisimulation equivalence is a congruence for all operators defined in P.*

Given the alternative definition of the probability obliterated bisimulation provided by Lemma 4, we will now consider simpler definitions for the quantitative premises for the rule format associated to this relation. Thus, we consider quantitative premises of the form $\theta(\{y\}) \trianglerighteq p$ rather than $\theta(Y) \trianglerighteq p$.

Taking $\mathfrak{t}_3$ and $\mathfrak{t}_4$ as before, we have that $\mathfrak{t}_3 \sim_o \mathfrak{t}_4$. The same example of the unary operator f, whose semantics is defined with a conveniently modify rule (1), shows that $f(\mathfrak{t}_3) \not\sim_a f(\mathfrak{t}_4)$ and hence the need that the quantitative premises can only be tested against 0.

Let $\mathfrak{t}_5 = a.(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0})$ and $\mathfrak{t}_6 = a.\boldsymbol{b.0} + a.\boldsymbol{c.0}$, and observe that $\mathfrak{t}_5 \sim_o \mathfrak{t}_6$. Take rule (2) as the semantic definition for f. Notice that $f(\mathfrak{t}_5) \xrightarrow{a} \boldsymbol{a.}(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0})$ is the only transition for $f(\mathfrak{t}_5)$, while the only possible transitions for $f(\mathfrak{t}_6)$ are $f(\mathfrak{t}_6) \xrightarrow{a} \boldsymbol{a.b.0}$ and $f(\mathfrak{t}_6) \xrightarrow{a} \boldsymbol{a.c.0}$. Since $a.\boldsymbol{b.0} \not\sim_o a.(\boldsymbol{b.0} \oplus_{0.5} \boldsymbol{c.0}) \not\sim_o a.\boldsymbol{c.0}$, $f(\mathfrak{t}_5) \not\sim_o f(\mathfrak{t}_6)$. Like for the convex bisimulation case, this shows that the target of a positive premise cannot appear in a d-sorted position of a subterm in the target of the conclusion.

Suppose now that the semantics of f is defined with the rule

$$\frac{x \xrightarrow{a} \mu \qquad \mu(\{y_1\}) > 0 \qquad \mu(\{y_2\}) > 0 \qquad y_1 \xrightarrow{b} \mu_1 \qquad y_2 \xrightarrow{c} \mu_2}{f(x) \xrightarrow{a} \mathbf{0}} \tag{5}$$

Notice that $f(\mathfrak{t}_5) \not\sim_o f(\mathfrak{t}_6)$ since $f(\mathfrak{t}_5) \xrightarrow{a} \mathbf{0}$ while $f(\mathfrak{t}_6)$ cannot perform any transition. This is due to the fact that, by allowing the same distribution variable μ to occur in different quantitative premises, we gain some knowledge of the structure of (the instance of) μ, in particular of its support set.

Consider now that f is defined with the left rule in (3) and g with an appropriate modification of the right rule in (3). Notice that $f(\mathtt{t}_5) \xrightarrow{a} \mathbf{0}$ but $f(\mathtt{t}_6)$ cannot perform any transition. Thus $f(\mathtt{t}_5) \not\sim_o f(\mathtt{t}_6)$. In this case, we are also gaining knowledge of the support set of μ, but this time through the rule associated to the operator g. Therefore we require that a target of a positive premise does not appear in the source of a positive or negative premise.

Consider now the rules

$$\frac{x \xrightarrow{a} \mu \quad \mu(\{y\}) > 0 \quad y \xrightarrow{b} \mu'}{f(x) \xrightarrow{a} \boldsymbol{g}(\mu)} \qquad \frac{x \xrightarrow{c} \mu}{g(x) \xrightarrow{c} \mathbf{0}} \tag{6}$$

Notice that the only transition for $f(\mathtt{t}_5)$ is $f(\mathtt{t}_5) \xrightarrow{a} \boldsymbol{g}(\boldsymbol{b}.\mathbf{0} \oplus_{0.5} \boldsymbol{c}.\mathbf{0})$ and the only transition for $f(\mathtt{t}_6)$ is $f(\mathtt{t}_6) \xrightarrow{a} \boldsymbol{g}(\boldsymbol{b}.\mathbf{0})$. Then $f(\mathtt{t}_5) \overset{a}{\rightsquigarrow} g(c.\mathbf{0}) \overset{c}{\rightsquigarrow} 0$ while $f(\mathtt{t}_6) \overset{a}{\rightsquigarrow} g(b.\mathbf{0})$ is the only possible "obliterated" transition for $f(\mathtt{t}_6)$. Then $f(\mathtt{t}_5) \not\sim_o f(\mathtt{t}_6)$. This is an alternative way of gaining information on the support set of a possible instance of μ in (6): on the one hand, by the quantitative premise on the first rule, we deduce that such instance has an element in the support set that performs a b-transition and, on the other hand, by having μ as an argument in the target of the conclusion, we may gather extra information from the same instance of μ through the rules for the semantics of the target of the conclusion (in this case, that μ has another element in the support set that performs a c-transition.) Therefore, we forbid that the target of a positive premise is both tested in a quantitative premise and used in the target of the conclusion.

Notice that the example in rules (4) also apply for probability obliterated bisimulation since $\mathtt{t}_1 \sim_o \mathtt{t}_2$ but $f(\mathtt{t}_1) \not\sim_o f(\mathtt{t}_2)$ with exactly the same explanation. Thus, we also request that the target of the conclusion is linear for all distribution variables on targets of positive premises.

Finally, consider a modification (4) where the left rule is instead

$$\frac{x \xrightarrow{a} \mu \quad \boldsymbol{g}(\mu,\mu)(\{y\}) > 0 \quad y \xrightarrow{a} \mu'}{f(x) \xrightarrow{a} \mathbf{0}} \tag{7}$$

It should not be difficult to observe that $f(\mathtt{t}_5) \xrightarrow{a} \mathbf{0}$ but $f(\mathtt{t}_6)$ cannot perform any transition. Thus $f(\mathtt{t}_5) \not\sim_o f(\mathtt{t}_6)$. For this reason we also require that the quantitative premises only allow linear distribution terms.

Definition 15. *Let $P = (\Sigma, A, R)$ be a well-founded PTSS. A rule $r \in R$ is in* probability obliterated *ntμfθ* format *if it has the form*

$$\frac{\bigcup_{m\in M}\{t_m \xrightarrow{a_m} \mu_m\} \quad \cup \quad \bigcup_{n\in N}\{t_n \xrightarrow{b_n}\!\!\!\!\!\!/\ \ \} \quad \cup \quad \bigcup_{l\in L}\{\theta_l(\{y_l\}) > 0\}}{f(\zeta_1,\ldots,\zeta_{\mathsf{rk}(f)}) \xrightarrow{a} \theta}$$

where all variables $\zeta_1,\ldots,\zeta_{\mathsf{rk}(f)}$, μ_m, with $m \in M$, and y_l, with $l \in L$, are different and the following restrictions are satisfied:

1. *For all $m \in M$, $\mathcal{V}(t_m) \cap \{\mu_{m'} \mid m' \in M\} = \emptyset$. Similarly, for all $n \in N$, $\mathcal{V}(t_n) \cap \{\mu_{m'} \mid m' \in M\} = \emptyset$.*
2. *For all $l \in L$, θ_l is linear for $\{\mu_{m'} \mid m' \in M\}$ and, moreover, for all $l, l' \in L$ with $l \neq l'$, $\mathcal{V}(\theta_l) \cap \mathcal{V}(\theta_{l'}) \cap \{\mu_m \mid m \in M\} = \emptyset$.*
3. *θ is linear for $\{\mu_{m'} \mid m' \in M\}$, $\mathcal{V}(\theta) \cap \big(\bigcup_{l\in L} \mathcal{V}(\theta_l)\big) \cap \{\mu_m \mid m \in M\} = \emptyset$, and no variable μ_m appear in a d-sorted position of a subterm of the target of the conclusion θ.*

A rule is in probability obliterated *ntμxθ* format *if its form is like above but has a conclusion of the form $x \xrightarrow{a} \theta$. P is in* probability obliterated *ntμfθ/ntμxθ* format *if all its rules are in probability obliterated ntμfθ/ntμxθ format.*

Condition 1 limits the form to exclude rules like the one on the left of (3). Condition 2 requires that the distribution terms on the quantitative premises are linear (excluding (6)), and that they do not share distributions variables on the target of positive premises (excluding (5)). Finally, condition 3 request that the target of the conclusion is linear (excluding (4)) and does not have targets of positive premises on d-sorted positions (excluding (2)) nor if they are used in quantitative premises (excluding (6)).

Finally, we state the congruence theorem for probability obliterated bisimulation equivalence.

Theorem 5. *Let P be a complete well-founded PTSS in probability obliterated $nt\mu f\theta/nt\mu x\theta$ format. Then, probability obliterated bisimulation equivalence is a congruence for all operators defined by P.*

6 Conclusion and Future Work

In this article, we presented three new rule formats that preserve three different bisimulation equivalences coarser than Larsen & Skou's bisimulation. These formats are more restricted variants of the $nt\mu f\theta/nt\mu x\theta$ format and notably, all of them can be seen as generalizations of the non-probabilistic *ntyft/ntyxt* format [7, 14]. For completeness we mention two other similar results on PTSSs that fall out of Larsen & Skou's bisimulation. They are [20], that presents a format for rooted branching bisimulation, and [26], that presents a format for non-expansiveness of ϵ-bisimulations.

Prior to the congruence theorems, we presented the different bisimulation equivalences, compare them, and, in particular, we gave a logic characterization for each of them. The intention of presenting these logic characterizations is to use them as the basis for the proof of full abstraction theorems (see, e.g., [8, 9, 14, 15].) Full abstraction theorems are somewhat dual to the congruence theorems. An equivalence relation is fully abstract with respect to a particular format and an equivalence relation $\equiv$ if it is the largest relation included in $\equiv$ that is a congruence for all operators definable by any PTSS in that format. In particular we are interested when $\equiv$ is the coarsest reasonable behavioral equivalence, namely, (possibilistic) trace equivalence. We are busy now on trying to prove this results for the formats presented here using the logic characterization as a means to construct the so called *testers*. As the current point of our investigation, we do not foresee major problems for all relations except for convex bisimulation equivalences, for which we may need to relax some of the conditions of the convex $nt\mu f\theta/nt\mu x\theta$ format.

References

[1] L. Aceto, W. Fokkink & C. Verhoef (2001): *Conservative Extension in Structural Operational Semantics*. In: *Current Trends in Theoretical Computer Science*, pp. 504–524, doi:10.1142/9789812810403_0004.

[2] L. Aceto, W. Fokkink & C. Verhoef (2001): *Structural operational semantics*. In: *Handbook of Process Algebra*, Elsevier, pp. 197–292, doi:10.1016/B978-044482830-9/50021-7.

[3] C. Baier (1998): *On Algorithmics Verification Methods for Probabilistic Systems*. Habilitation thesis, University of Mannheim.

[4] F. Bartels (2002): *GSOS for Probabilistic Transition Systems*. *Electr. Notes Theor. Comput. Sci.* 65(1), pp. 29–53, doi:10.1016/S1571-0661(04)80358-X.

[5] F. Bartels (2004): *On Generalised Coinduction and Probabilistic Specification Formats*. Ph.D. thesis, VU University Amsterdam.

[6] B. Bloom, S. Istrail & A.R. Meyer (1995): *Bisimulation can't be traced*. *J. ACM* 42, pp. 232–268, doi:10.1145/200836.200876.

[7] R. Bol & J.F. Groote (1996): *The meaning of negative premises in transition system specifications*. *J. ACM* 43, pp. 863–914, doi:10.1145/234752.234756.

[8] P.R. D'Argenio, D. Gebler & M.D. Lee (2015): *A general SOS theory for the specification of probabilistic transition systems*. Available at `http://www.cs.famaf.unc.edu.ar/~lee/publications/DArgenioGeblerLee.pdf`. Submitted.

[9] P.R. D'Argenio & M.D. Lee (2012): *Probabilistic Transition System Specification: Congruence and Full Abstraction of Bisimulation*. In: *Proc. FoSSaCS'12*, *LNCS* 7213, Springer, pp. 452–466, doi:10.1007/978-3-642-28729-9_30.

[10] P.R. D'Argenio, P. Sánchez Terraf & N. Wolovick (2012): *Bisimulations for non-deterministic labelled Markov processes*. *Mathematical Structures in Computer Science* 22(1), pp. 43–68, doi:10.1017/S0960129511000454.

[11] W. Fokkink (2000): *Rooted Branching Bisimulation as a Congruence*. *J. Comput. Syst. Sci.* 60(1), pp. 13–37, doi:10.1006/jcss.1999.1663.

[12] W. Fokkink & C. Verhoef (1998): *A Conservative Look at Operational Semantics with Variable Binding*. *Inf. Comput.* 146(1), pp. 24–54, doi:10.1006/inco.1998.2729.

[13] R.J. van Glabbeek (2004): *The meaning of negative premises in transition system specifications II*. *J. Log. Algebr. Program.* 60-61, pp. 229–258, doi:10.1016/j.jlap.2004.03.007.

[14] J.F. Groote (1993): *Transition system specifications with negative premises*. *Theoretical Computer Science* 118(2), pp. 263–299, doi:10.1016/0304-3975(93)90111-6.

[15] J.F. Groote & F.W. Vaandrager (1992): *Structured Operational Semantics and Bisimulation as a Congruence*. *Inf. Comput.* 100(2), pp. 202–260, doi:10.1016/0890-5401(92)90013-6.

[16] H. Hermanns, A. Parma, R. Segala, B. Wachter & L. Zhang (2011): *Probabilistic Logical Characterization*. *Inf. Comput.* 209(2), pp. 154–172, doi:10.1016/j.ic.2010.11.024.

[17] R. Lanotte & S. Tini (2005): *Probabilistic Congruence for Semistochastic Generative Processes*. In: *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005.*, pp. 63–78, doi:10.1007/978-3-540-31982-5_4.

[18] R. Lanotte & S. Tini (2009): *Probabilistic bisimulation as a congruence*. *ACM Trans. Comput. Log.* 10(2), doi:10.1145/1462179.1462181.

[19] K.G. Larsen & A/ Skou (1991): *Bisimulation through Probabilistic Testing*. *Inf. Comput.* 94(1), pp. 1–28, doi:10.1016/0890-5401(91)90030-6.

[20] M.D. Lee & E. de Vink (2015): *Rooted branching bisimulation as a congruence for probabilistic transition systems*. In: *Proc. QAPL'15*, EPTCS. To appear.

[21] M.D. Lee, D. Gebler & P.R. D'Argenio (2012): *Tree Rules in Probabilistic Transition System Specifications with Negative and Quantitative Premises*. In: *Proc. EXPRESS/SOS'12*, *EPTCS* 89, pp. 115–130, doi:10.4204/EPTCS.89.9.

[22] R. Milner (1989): *Communication and Concurrency*. Prentice-Hall.

[23] M.R. Mousavi, M.A. Reniers & J.F. Groote (2007): *SOS formats and meta-theory: 20 years after*. *Theor. Comput. Sci.* 373(3), pp. 238–272, doi:10.1016/j.tcs.2006.12.019.

[24] G. Plotkin (1981): *A structural approach to operational semantics*. Report DAIMI FN-19, Aarhus University, doi:doi:10.1016/j.jlap.2004.05.001. Reprinted in *J. Log. Algebr. Program.*, 60-61:17-139, 2004.

[25] R. Segala (1995): *Modeling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. thesis, MIT.

[26] S. Tini (2010): *Non-expansive ϵ-bisimulations for probabilistic processes*. *Theor. Comput. Sci.* 411(22-24), pp. 2202–2222, doi:10.1016/j.tcs.2010.01.027.

Analysing and Comparing Encodability Criteria

Kirstin Peters*
TU Dresden, Germany

Rob van Glabbeek
NICTA†, Sydney, Australia
Computer Science and Engineering, UNSW, Sydney, Australia

Encodings or the proof of their absence are the main way to compare process calculi. To analyse the quality of encodings and to rule out trivial or meaningless encodings, they are augmented with quality criteria. There exists a bunch of different criteria and different variants of criteria in order to reason in different settings. This leads to incomparable results. Moreover it is not always clear whether the criteria used to obtain a result in a particular setting do indeed fit to this setting. We show how to formally reason about and compare encodability criteria by mapping them on requirements on a relation between source and target terms that is induced by the encoding function. In particular we analyse the common criteria *full abstraction*, *operational correspondence*, *divergence reflection*, *success sensitiveness*, and *respect of barbs*; e.g. we analyse the exact nature of the simulation relation (coupled simulation versus bisimulation) that is induced by different variants of operational correspondence. This way we reduce the problem of analysing or comparing encodability criteria to the better understood problem of comparing relations on processes.

1 Introduction

Encodings are used to compare process calculi and to reason about their expressive power. Encodability criteria are conditions that limit the existence of encodings. Their main purpose is to rule out trivial or meaningless encodings, but they can also be used to limit attention to encodings that are of special interest in a particular domain or for a particular purpose. These quality criteria are the main tool in *separation results*, saying that one calculus is not expressible in another one; here one has to show that no encoding meeting these criteria exists. To obtain stronger separation results, care has to be taken in selecting quality criteria that are not too restrictive. For *encodability results*, saying that one calculus is expressible in another one, all one needs is an encoding, together with criteria testifying for the quality of the encoding. Here it is important that the criteria are not too weak.

In the literature various different criteria and different variants of the same criteria are employed to achieve separation and encodability results [6, 13, 15, 17, 18, 14, 3, 19, 2, 8, 24, 7]. Some criteria, like full abstraction or operational correspondence, are used frequently. Other criteria are used to enforce a property of encodings that might only be necessary within a certain domain. For instance, the homomorphic translation of the parallel operator—in general a rather strict criterion—was used in [17] to show the absence of an encoding from the synchronous into the asynchronous π-calculus, because this requirement forbids for the introduction of global coordinators. Thus this criterion is useful when reasoning about the concurrent behaviour of processes, although it is in general too strict to reason about their interleaving behaviour. Unfortunately it is not always obvious or clear whether the criteria used to obtain a result in a particular setting do indeed fit to this setting. Indeed, as discussed in [24], the

*Supported by funding of the Excellence Initiative by the German Federal and State Governments (Institutional Strategy, measure 'support the best').

†NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

S. Crafa and D. Gebler (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2015)
EPTCS 190, 2015, pp. 46–60, doi:10.4204/EPTCS.190.4

© K. Peters & R. J. van Glabbeek
This work is licensed under the Creative Commons Attribution License.

homomorphic translation of the parallel operator forbids more than global coordinators, i.e., is too strict even in a concurrent setting.

The different purposes of encodability criteria lead to very different kinds of conditions that are usually hard to analyse and compare directly. In fact even widely used criteria—as full abstraction—seem not to be fully understood by the community, as the need for articles as [9, 20] shows. In contrast to that, relations on processes—such as simulations and bisimulations—are a very well studied and understood topic (see for example [5]). Moreover it is natural to describe the behaviour of terms, or compare them, modulo some equivalence relation. Also many encodability criteria, like operational correspondence, are obviously designed with a particular kind of relation between processes in mind. Therefore, in order to be able to formally reason about encodability criteria, to completely capture and describe their semantic effect, and to analyse side conditions of combinations of criteria, we map them on conditions on relations between source and target terms.

We consider the disjoint union $\mathscr{P}_S \uplus \mathscr{P}_T$ of the terms or processes from the source and target languages of an encoding. Then we describe the effect an encodability criterion **C** has on the class of permitted encoding functions in terms of a relation $\mathscr{R}_{[\![\cdot]\!]}$ that relates at least all source terms to their literal translations, i.e., contains the pair $(S, [\![S]\!])$ for all source terms S. If the encodability criterion **C** is defined w.r.t. some additional relations on the source or target languages, as it is the case for full abstraction and operational correspondence, we usually also include these relations in $\mathscr{R}_{[\![\cdot]\!]}$. In order to completely capture the effect of a criterion **C** we aim at iff-results of the form

$$[\![\cdot]\!] \text{ satisfies } \mathbf{C} \text{ iff there exists a relation } \mathscr{R}_{[\![\cdot]\!]} \text{ such that } \forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]} \text{ and } \mathsf{P}\big(\mathscr{R}_{[\![\cdot]\!]}\big),$$

where P is the condition that captures the effect of **C**. For example, an encoding reflects divergence iff there exists a relation $\mathscr{R}_{[\![\cdot]\!]}$ such that $\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$ and $\mathscr{R}_{[\![\cdot]\!]}$ reflects divergence.

We illustrate this approach by applying it to some very common criteria. We start with divergence reflection in §3.1, because it is simple and well understood. Accordingly, we do not gain significant new insights, but it suits us very well to introduce our approach. In the same way success sensitiveness and respect of barbs are analysed. We then switch to the criteria full abstraction in §3.2 and operational correspondence in §3.3, which are possibly not completely understood yet. In particular, we show a connection between full abstraction and transitivity, and prove to which kinds of simulation relations common variants of operational correspondence are linked. In §4 we analyse the effects of combining the above criteria. Since we first map the criteria to conditions on relations between source and target terms, analysing their combined effect requires us to identify a suitable witness relation for the combined conditions. Combining divergence reflection and success sensitiveness is simple, as illustrated in §4.1. Combining these two criteria with operational correspondence (§4.2) is more elaborate. Finally we analyse the effect of combining full abstraction with operational correspondence in §4.3.

All claims in this paper have been proved using the interactive theorem prover Isabelle/HOL [16]. The Isabelle implementation of the theories is available in the 'Archive of Formal Proofs' at

`http://afp.sourceforge.net/entries/Encodability_Process_Calculi.shtml`.

2 Technical Preliminaries

We analyse criteria used to reason about the quality of encodings between process calculi. We do not force any limitations on the considered calculi. A *process calculus* is a language $\mathscr{L}_C = (\mathscr{P}_C, \longmapsto_C)$ consisting of a set of terms $\mathscr{P}_C$—its *syntax*—and a relation on terms $\longmapsto_C \subseteq \mathscr{P}_C \times \mathscr{P}_C$—its *semantics*. The elements of $\mathscr{P}_C$ are called *process terms* or shortly processes or terms.

Here we assume that the semantics of the language is provided as a so-called reduction semantics, because in the context of encodings the treatment of reductions is simpler—the consideration of labelled semantics and of criteria using labelled steps is left for further work. A *step* $P \longmapsto_C P'$ is an element $(P,P') \in \longmapsto_C$. Let $\Longmapsto_C$ denote the reflexive and transitive closure of $\longmapsto_C$. We write $P \longmapsto_C$ if $\exists P'.\ P \longmapsto_C P'$ and $P \longmapsto_C{}^{\omega}$ if P can do an infinite sequence of steps. A term P such that $P \longmapsto_C{}^{\omega}$ is called *divergent*.

Languages can be augmented with (a set of) relations $\mathscr{R}_C \subseteq \mathscr{P}_C^2$ on their processes. If $\mathscr{R} \subseteq B^2$ is a relation and $B' \subseteq B$, then $\mathscr{R}\restriction_{B'} = \{\,(x,y) \mid x,y \in B' \wedge (x,y) \in \mathscr{R}\,\}$ denotes the restriction of $\mathscr{R}$ to the domain B'. A relation $\mathscr{R}$ *preserves* some condition $\mathsf{P} : B \to \mathbb{B}$ (with $\mathbb{B}$ representing the Booleans) if whenever $(P,Q) \in \mathscr{R}$ and P satisfies P then Q satisfies P. A relation $\mathscr{R}$ *reflects* P if whenever $(P,Q) \in \mathscr{R}$ and Q satisfies P then also P satisfies P. Finally $\mathscr{R}$ *respects* a condition P if $\mathscr{R}$ preserves and reflects it. We use $\mathrm{r}(\cdot)$, $\mathrm{s}(\cdot)$, and $\mathrm{t}(\cdot)$ to denote the *reflexive*, *symmetric*, and *transitive closure* of a binary relation, respectively.

Relations on process terms are an important tool to reason about processes and languages. Of special interest are simulation relations; in particular bisimulations. $\mathscr{R}$ is a bisimulation if any two related processes mutually simulate their respective sequences of steps, such that the derivatives are again related.

Definition 2.1 (Bisimulation) *$\mathscr{R}$ is a* (weak reduction) bisimulation *if for each $(P,Q) \in \mathscr{R}$:*

- *$P \Longmapsto P'$ implies $\exists Q'.\ Q \Longmapsto Q' \wedge (P',Q') \in \mathscr{R}$*
- *$Q \Longmapsto Q'$ implies $\exists P'.\ P \Longmapsto P' \wedge (P',Q') \in \mathscr{R}$*

Two terms are bisimilar *if there exists a bisimulation that relates them.*

The definition of a *strong (reduction) bisimulation* is obtained by replacing all $\Longmapsto$ by $\longmapsto$ in the above definition, i.e., a strong bisimulation requires that a step has to be simulated by a single step. Coupled similarity is strictly weaker than bisimilarity. As pointed out in [21], in contrast to bisimilarity it allows for intermediate states in simulations: states that cannot be identified with states of the simulated term. Each symmetric coupled simulation is a bisimulation.

Definition 2.2 (Coupled Simulation) *A relation $\mathscr{R}$ is a* (weak reduction) coupled simulation *if both $(\exists Q'.\ Q \Longmapsto Q' \wedge (P',Q') \in \mathscr{R})$ and $(\exists Q'.\ Q \Longmapsto Q' \wedge (Q',P') \in \mathscr{R})$ whenever $(P,Q) \in \mathscr{R}$ and $P \Longmapsto P'$. Two terms are* coupled similar *if they are related by a coupled simulation in both directions.*

An *encoding* from $\mathscr{L}_{\mathrm{S}} = (\mathscr{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}})$ into $\mathscr{L}_{\mathrm{T}} = (\mathscr{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}})$ relates two process calculi. We call $\mathscr{L}_{\mathrm{S}}$ the *source* and $\mathscr{L}_{\mathrm{T}}$ the *target language*. Accordingly, terms of $\mathscr{P}_{\mathrm{S}}$ are *source terms* and of $\mathscr{P}_{\mathrm{T}}$ *target terms*. In the simplest case an encoding from $\mathscr{L}_{\mathrm{S}}$ into $\mathscr{L}_{\mathrm{T}}$ is an *encoding function* $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ from source terms into target terms. Sometimes an encoding is defined by several functions, such as the encoding function and the renaming policy used in the framework of [8]. Else we identify an encoding with its encoding function.

An *encodability criterion* is a predicate on encoding functions, used to reason about the quality of encodings. We analyse such criteria by mapping them on requirements on relations $\mathscr{R}_{[\![\cdot]\!]} \subseteq (\mathscr{P}_{\mathrm{S}} \uplus \mathscr{P}_{\mathrm{T}})^2$ on the disjoint union of the source and target terms of the considered encodings $[\![\cdot]\!]$. To simplify the presentation we assume henceforth that $\mathscr{P}_{\mathrm{S}} \cap \mathscr{P}_{\mathrm{T}} = \emptyset$ and thus $\mathscr{P}_{\mathrm{S}} \uplus \mathscr{P}_{\mathrm{T}} = \mathscr{P}_{\mathrm{S}} \cup \mathscr{P}_{\mathrm{T}}$. The Isabelle proofs do not rely on such an assumption. We say that a condition $\mathsf{P} : (\mathscr{P}_{\mathrm{S}} \uplus \mathscr{P}_{\mathrm{T}}) \to \mathbb{B}$ is *preserved* by an encoding if for all source terms S that satisfy P, the condition P also holds for $[\![S]\!]$. A condition is *reflected* by an encoding if whenever $[\![S]\!]$ satisfies it, then so does S. Finally an encoding *respects* a condition if it both preserves and reflects it.

3 Analysing Encodability Criteria

An encoding function $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ maps source terms on target terms. Thereby it induces a relation on the combined domain of source and target terms that relates source terms with their literal translations. We start with this relation, i.e., in the simplest case we map an encodability criterion to a requirement on a relation $\mathscr{R}_{[\![\cdot]\!]} \subseteq (\mathscr{P}_S \uplus \mathscr{P}_T)^2$ that contains at least the pairs $(S, [\![S]\!])$ for all source terms $S \in \mathscr{P}_S$. If we consider a criterion that is defined w.r.t. some relations on the source or target, we will also include these relations in $\mathscr{R}_{[\![\cdot]\!]}$, possibly closing the latter under reflexivity, symmetry, and/or transitivity.

Alternatively, we could require that $\mathscr{R}_{[\![\cdot]\!]}$ relates source terms and their literal translations in both directions, meaning that $(S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$ and $([\![S]\!], S) \in \mathscr{R}_{[\![\cdot]\!]}$ for all source terms $S \in \mathscr{P}_S$. However, this condition limits our analysis to properties that are respected. It does not allow us to reason about properties like divergence reflection, where some condition need only to be reflected but not necessarily be preserved, or vice versa. Accordingly we follow the first approach.

3.1 Divergence Reflection and Observables

We start with divergence reflection as defined in [8], because it is often easy to establish and well understood. An encoding reflects divergence if it does not introduce divergence, i.e., if all divergent translations result from divergent source terms.

Definition 3.1 (Divergence Reflection) *An encoding* $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ reflects divergence *if* $[\![S]\!] \longmapsto_T{}^{\omega}$ *implies* $S \longmapsto_S{}^{\omega}$ *for all source terms* $S \in \mathscr{P}_S$.

We can reformulate this criterion as follows: An encoding reflects divergence if it reflects the predicate $\lambda P.\ P \longmapsto^{\omega}$. To analyse this criterion it suffices to consider the relation $\{ (S, [\![S]\!]) \mid S \in \mathscr{P}_S \}$. It is obvious that an encoding reflects divergence iff $\{ (S, [\![S]\!]) \mid S \in \mathscr{P}_S \}$ reflects divergence, i.e., reflects the predicate $\lambda P.\ P \longmapsto^{\omega}$. In fact we can generalise this case. If an encodability criterion can be described by the preservation or reflection of a predicate, then an encoding satisfies this criterion iff $\{ (S, [\![S]\!]) \mid S \in \mathscr{P}_S \}$ preserves or reflects this predicate. Of course direction "if" holds for any relation that contains at least the pairs $(S, [\![S]\!])$. We use the relation $\{ (S, [\![S]\!]) \mid S \in \mathscr{P}_S \}$ as a witness and it allows us to analyse the combination of different criteria later.

Lemma 3.2 (Preservation) *Let* $\mathsf{P} : (\mathscr{P}_S \uplus \mathscr{P}_T) \to \mathbb{B}$ *be a predicate. An encoding preserves the predicate* P *iff* $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_{[\![\cdot]\!]}$ *preserves* P.

We obtain a similar result if we replace the unary predicate $\mathsf{P}(\cdot)$ by the binary predicate $\mathsf{P}(\cdot,\cdot)$ of type $(\mathscr{P}_S \uplus \mathscr{P}_T) \times \mathscr{T} \to \mathbb{B}$ for some arbitrary type $\mathscr{T}$ to represent predicates with several parameters. Moreover we obtain the same result for either reflection or respect instead of preservation.

Accordingly an encoding reflects divergence, i.e., the predicate $\lambda P.\ P \longmapsto^{\omega}$, iff there exists a relation $\mathscr{R}_{[\![\cdot]\!]}$ that relates at least each source term to its literal translation and reflects this predicate.

Lemma 3.3 (Divergence Reflection) *An encoding* $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ *from* $\mathscr{L}_S$ *into* $\mathscr{L}_T$ *reflects divergence iff* $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_{[\![\cdot]\!]}$ *reflects divergence.*

In a similar way we can deal with the criterion barb sensitiveness. A *barb* is a property of a process that is treated as an observable, and whose reachability should be respected by an encoding. We assume that $\mathscr{B}$ is a set of barbs that contains at least all barbs of the source and the target language. Moreover we assume that each language $\mathscr{L}$ specifies its own predicate $\cdot \downarrow_{\mathscr{L}} \cdot$ such that $P \downarrow_{\mathscr{L}} a$ returns true if $P \in \mathscr{P}_{\mathscr{L}}$

and P has the barb a in $\mathscr{L}$. If a barb a is not relevant or present in a language $\mathscr{L}$ then $P\downarrow_{\mathscr{L}} a$ does not hold for any $P \in \mathscr{P}_{\mathscr{L}}$. We use $P\Downarrow_{\mathscr{L}} a$ if P reaches the barb a in $\mathscr{L}$, i.e., $P\Downarrow_{\mathscr{L}} a \triangleq \exists P'.P \Longrightarrow_{\mathscr{L}} P' \wedge P'\downarrow_{\mathscr{L}} a$.

An encoding weakly respects source term barbs iff it respects the predicate $\lambda P\, a.\, P\Downarrow a$. This holds iff $\{\,(S,[\![S]\!]) \mid S \in \mathscr{P}_{\mathrm{S}}\,\}$ respects this predicate, which in turn is the case iff there exists a relation $\mathscr{R}_{[\![\cdot]\!]}$ that relates at least each source term to its literal translation and respects this predicate.

Lemma 3.4 (Barb Sensitiveness) *Assume $\mathscr{L}_{\mathrm{S}}$ and $\mathscr{L}_{\mathrm{T}}$ each define a predicate $\cdot\downarrow\cdot : \mathscr{P}\times\mathscr{B}\to\mathbb{B}$. $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}}\to\mathscr{P}_{\mathrm{T}}$ weakly respects barbs iff $\exists\mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S,[\![S]\!])\in\mathscr{R}_{[\![\cdot]\!]}\big)\wedge\mathscr{R}_{[\![\cdot]\!]}$ weakly respects barbs.*

Again we obtain a similar result if we replace respect by preservation or reflection or if we consider the existence instead of the reachability of barbs.

However, only very few encodings directly preserve or reflect barbs. More often barbs are translated, as for example in the encodings between different variants of the π-calculus in [15, 23] or the two translations from CSP into variants of CCS with name passing in [10]. Since we do not fix the definition of $\cdot\downarrow_{\mathscr{L}}\cdot$, this can for instance be expressed by adapting this predicate in the target language.

In a similar way we can deal with the criterion success sensitiveness. This criterion was proposed by Gorla as part of his encodability framework [8]. An encoding is success sensitive if it respects reachability of a particular process $\checkmark$ that represents successful termination, or some other form of success, and is added to the syntax of the source as well as the target language. We write $P\downarrow_{\checkmark}$ to denote the fact that P is successful—however this predicate might be defined in the particular source or target language. Reachability of success is then defined as $P\Downarrow_{\checkmark} \triangleq \exists P'.\, P\Longrightarrow P'\wedge P'\downarrow_{\checkmark}$. An encoding is success sensitive if each source term and its translation answer the test for reachability of success in the same way.

Definition 3.5 (Success Sensitiveness) *Let $\mathscr{L}_{\mathrm{S}}$ and $\mathscr{L}_{\mathrm{T}}$ each define a predicate $\cdot\downarrow_{\checkmark}\colon\mathscr{P}\to\mathbb{B}$. An encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}}\to\mathscr{P}_{\mathrm{T}}$ is* success sensitive *if, for all $S\in\mathscr{P}_{\mathrm{S}}$, $S\Downarrow_{\checkmark}$ iff $[\![S]\!]\Downarrow_{\checkmark}$.*

Accordingly, an encoding is success sensitive iff it respects the predicate $\lambda P.\, P\Downarrow_{\checkmark}$. This is the case iff $\{\,(S,[\![S]\!]) \mid S\in\mathscr{P}_{\mathrm{S}}\,\}$ respects this predicate, which in turn is the case iff there exists a relation $\mathscr{R}_{[\![\cdot]\!]}$ that relates at least each source term to its literal translation and respects this predicate.

Lemma 3.6 (Success Sensitiveness) *Assume $\mathscr{L}_{\mathrm{S}}$ and $\mathscr{L}_{\mathrm{T}}$ each define a predicate $\cdot\downarrow_{\checkmark}\colon\mathscr{P}\to\mathbb{B}$. An encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}}\to\mathscr{P}_{\mathrm{T}}$ is success sensitive iff $\exists\mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S,[\![S]\!])\in\mathscr{R}_{[\![\cdot]\!]}\big)\wedge\mathscr{R}_{[\![\cdot]\!]}$ respects $\lambda P.\, P\Downarrow_{\checkmark}$.*

Success sensitiveness links source term behaviours to behaviours of target terms. If the source and the target language are very different, they can impose quite different kinds of behaviour that might be hard to compare directly. For example, observables in the π-calculus refer to the existence of unguarded input or output prefixes [11], whereas in the core of mobile ambients there are no in- or outputs but only ambients and action prefixes that describe the entering, leaving, and opening of an ambient [4]. Success sensitiveness allows to compare such languages by introducing a new kind of barb that can be understood in both calculi. If we want to compare two languages that are very similar, such as two variants of the same calculus, we can demand stricter encodability criteria and compare their barbs directly.

Next we concentrate on criteria that cannot be expressed simply by the preservation or reflection of some predicate.

3.2 Full Abstraction

Full abstraction was probably the first criterion that was widely used to reason about the quality of encodings [26, 12, 22]. This criterion is defined w.r.t. a relation $\mathscr{R}_{\mathrm{S}}\subseteq\mathscr{P}_{\mathrm{S}}^2$ on source terms and a relation $\mathscr{R}_{\mathrm{T}}\subseteq\mathscr{P}_{\mathrm{T}}^2$ on target terms. An encoding is fully abstract w.r.t. $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$ if two source terms are related by $\mathscr{R}_{\mathrm{S}}$ iff their literal translations are related by $\mathscr{R}_{\mathrm{T}}$.

Definition 3.7 (Full Abstraction) *An encoding* $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ *is* fully abstract *w.r.t. the relations* $\mathscr{R}_S \subseteq \mathscr{P}_S^2$ *and* $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ *if, for all* $S_1, S_2 \in \mathscr{P}_S$, $(S_1, S_2) \in \mathscr{R}_S$ *iff* $([\![S_1]\!], [\![S_2]\!]) \in \mathscr{R}_T$.

There are a number of trivial full abstraction results, i.e., results that hold for all (or nearly all) encodings (see e.g. [9, 20]). In particular, for each encoding and each target term relation $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ there exits a source term relation $\mathscr{R}_S \subseteq \mathscr{P}_S^2$, namely $\{ (S_1, S_2) \mid ([\![S_1]\!], [\![S_2]\!]) \in \mathscr{R}_T \}$, such that the encoding is fully abstract w.r.t. $\mathscr{R}_S$ and $\mathscr{R}_T$. For each injective encoding and each source term relation $\mathscr{R}_S \subseteq \mathscr{P}_S^2$, there exits $\mathscr{R}_T \subseteq \mathscr{P}_T^2$, namely $\{ ([\![S_1]\!], [\![S_2]\!]) \mid (S_1, S_2) \in \mathscr{R}_S \}$, such that the encoding is fully abstract w.r.t. $\mathscr{R}_S$ and $\mathscr{R}_T$. Accordingly we consider full abstraction w.r.t. fixed source and target term relations.

As suggested above, we map this criterion on a relation that relates at least each source term to its literal translation and includes the relations $\mathscr{R}_S$ and $\mathscr{R}_T$. If we additionally add pairs of the form $([\![S]\!], S)$ for all $S \in \mathscr{P}_S$, we make an interesting observation. If we surround the pair $(S_1, S_2) \in \mathscr{R}_S$ by the pairs $([\![S_1]\!], S_1)$ and $(S_2, [\![S_2]\!])$ and add transitivity we obtain the pair $([\![S_1]\!], [\![S_2]\!])$. Similarly, from transitivity, $(S_1, [\![S_1]\!])$, $([\![S_1]\!], [\![S_2]\!])$, and $([\![S_2]\!], S_2)$ we obtain the pair (S_1, S_2). Because of this, an encoding is fully abstract w.r.t. the preorders $\mathscr{R}_S$ and $\mathscr{R}_T$ iff there exists a transitive relation $\mathscr{R}_{[\![\cdot]\!]}$ that relates at least each source term to its literal translation in both directions, such that the restriction of $\mathscr{R}_{[\![\cdot]\!]}$ to source/target terms is $\mathscr{R}_S$/$\mathscr{R}_T$.

Lemma 3.8 (Full Abstraction) $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ *is* fully abstract *w.r.t. the preorders* $\mathscr{R}_S \subseteq \mathscr{P}_S^2$ *and* $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ *iff* $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S, [\![S]\!]), ([\![S]\!], S) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_S = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_S} \wedge \mathscr{R}_T = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_T} \wedge \mathscr{R}_{[\![\cdot]\!]}$ *is transitive.*

Thus an encoding is fully abstract w.r.t. $\mathscr{R}_S$ and $\mathscr{R}_T$ if the encoding function combines the relations $\mathscr{R}_S$ and $\mathscr{R}_T$ in a transitive way.

In order to allow combinations with criteria like divergence reflection, i.e., predicates that are not respected but preserved or reflected, we get rid of the requirement on the pairs $([\![S]\!], S)$. Therefore we consider the symmetric closure of $\mathscr{R}_{[\![\cdot]\!]}$. An encoding is fully abstract w.r.t. the equivalences $\mathscr{R}_S$ and $\mathscr{R}_T$ iff there exists a relation $\mathscr{R}_{[\![\cdot]\!]}$ that relates at least each source term to its literal translation, such that the restriction of the symmetric closure of $\mathscr{R}_{[\![\cdot]\!]}$ to source/target terms is $\mathscr{R}_S$/$\mathscr{R}_T$ and the symmetric closure of $\mathscr{R}_{[\![\cdot]\!]}$ is a preorder.

Lemma 3.9 (Full Abstraction) *An encoding* $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ *is fully abstract w.r.t. the equivalences* $\mathscr{R}_S \subseteq \mathscr{P}_S^2$ *and* $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ *iff* $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_S = \mathrm{s}\big(\mathscr{R}_{[\![\cdot]\!]}\big) \restriction_{\mathscr{P}_S} \wedge \mathscr{R}_T = \mathrm{s}\big(\mathscr{R}_{[\![\cdot]\!]}\big) \restriction_{\mathscr{P}_T} \wedge \mathrm{s}\big(\mathscr{R}_{[\![\cdot]\!]}\big)$ *is a preorder.*

Since it is always possible to construct a relation that includes $\mathscr{R}_S$, $\mathscr{R}_T$, and pairs $(S, [\![S]\!])$, the crucial requirement on the right-hand side is transitivity. A discussion of this criterion and references to earlier such discussions can be found in [22, 9].

3.3 Operational Correspondence

To strengthen full abstraction it is often combined with operational correspondence. This criterion requires that source terms and their translations 'behave' similar, by requiring that steps are preserved and reflected modulo some target term relation $\mathscr{R}_T \subseteq \mathscr{P}_T^2$. Intuitively an encoding is operational corresponding w.r.t. $\mathscr{R}_T$ if each source term step is simulated by its translation, i.e., $[\![\cdot]\!]$ does not remove source behaviour (*completeness*), and each step of the target is part of the simulation of a source term step, i.e., $[\![\cdot]\!]$ does not introduce new behaviour (*soundness*). There are a number of different variants of this criterion. We consider three unlabelled variants [15, 8]. In particular the last variant, proposed in [8], was used for numerous encodability and separation results.

Definition 3.10 (Operational Correspondence) *An encoding* $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ *is* strongly operationally corresponding *w.r.t.* $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ *if it is:*
Strongly Complete: $\forall S, S'.\ S \longmapsto_S S'$ *implies* $(\exists T.\ [\![S]\!] \longmapsto_T T \wedge ([\![S']\!], T) \in \mathscr{R}_T)$
Strongly Sound: $\forall S, T.\ [\![S]\!] \longmapsto_T T$ *implies* $(\exists S'.\ S \longmapsto_S S' \wedge ([\![S']\!], T) \in \mathscr{R}_T)$
$[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ *is* operationally corresponding *w.r.t.* $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ *if it is:*
Complete: $\forall S, S'.\ S \Longmapsto_S S'$ *implies* $(\exists T.\ [\![S]\!] \Longmapsto_T T \wedge ([\![S']\!], T) \in \mathscr{R}_T)$
Sound: $\forall S, T.\ [\![S]\!] \Longmapsto_T T$ *implies* $(\exists S'.\ S \Longmapsto_S S' \wedge ([\![S']\!], T) \in \mathscr{R}_T)$
$[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ *is* weakly operationally corresponding *w.r.t.* $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ *if it is:*
Complete: $\forall S, S'.\ S \Longmapsto_S S'$ *implies* $(\exists T.\ [\![S]\!] \Longmapsto_T T \wedge ([\![S']\!], T) \in \mathscr{R}_T)$
Weakly Sound: $\forall S, T.\ [\![S]\!] \Longmapsto_T T$ *implies* $(\exists S', T'.\ S \Longmapsto_S S' \wedge T \Longmapsto_T T' \wedge ([\![S']\!], T') \in \mathscr{R}_T)$

Again this criterion is trivial if we do not fix the target term relation. Each encoding is operational corresponding w.r.t. the universal relation on target terms.

The formulation of operational correspondence (in all its variants) strongly reminds us of simulation relations on processes, such as bisimilarity. Obviously this criterion is designed in order to establish a simulation-like relation between source and target terms. We now determine the exact nature of this relation. The first two variants exactly describe strong and weak bisimilarity up to $\mathscr{R}_T$. More precisely, an encoding is operational corresponding w.r.t. a preorder $\mathscr{R}_T$ that is a bisimulation iff there exists a preorder $\mathscr{R}_{[\![\cdot]\!]}$, such as $t(r(\{(S, [\![S]\!]) \mid S \in \mathscr{P}_S\} \cup \mathscr{R}_T))$, that is a bisimulation, relates at least all source terms to their literal translations, and such that $\mathscr{R}_T = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_T}$, and for all pairs $(S,T) \in \mathscr{R}_{[\![\cdot]\!]}$ it holds that $([\![S]\!], T) \in \mathscr{R}_T$. The last condition is necessary to ensure operational correspondence, and $\mathscr{R}_T = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_T}$ ensures that $\mathscr{R}_T$ is a bisimulation if $\mathscr{R}_{[\![\cdot]\!]}$ is. Accordingly, operational correspondence ensures that source terms and their translations are bisimilar.

Lemma 3.11 (Operational Correspondence) *An encoding* $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ *is operational corresponding w.r.t. a preorder* $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ *that is a bisimulation iff* $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_T = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_T} \wedge \big(\forall S, T.\ (S,T) \in \mathscr{R}_{[\![\cdot]\!]} \to ([\![S]\!], T) \in \mathscr{R}_T\big) \wedge \mathscr{R}_{[\![\cdot]\!]}$ *is a preorder and a bisimulation.*

We obtain the same result if we replace operational correspondence by strong operational correspondence and bisimulation by strong bisimulation.

Weak and strong bisimilarity are often considered as *the* standard reference relations for calculi like the π-calculus. Thus the above result imposes an important property for the comparison of languages. If bisimilarity is the standard reference relation, i.e., if we usually do not record differences between terms that cannot be observed by bisimilarity, then an encoding that ensures that source terms and their translations are bisimilar strongly validates the claim that the target language is at least as expressive as the source language. Nonetheless, comparisons of different languages are very often considered only modulo weak operational correspondence and not operational correspondence. As discussed in [21, 10], relating source terms and their literal translations by a bisimulation does not allow for intermediate states, i.e., states that occur in simulations of source term steps and thus intuitively are in between two source term translations but are not related to source terms themselves. Intermediate states result from partial commitments. If a source term can evolve to one of three different derivatives, operational correspondence (in all variants) ensures that the translation has the same possible evolutions. But operational correspondence requires that the decision on which of the three possibilities is chosen is done in a single step. Weak operational correspondence allows for partial commitments, where a first step may rule out one possibility but not decide on one of the remaining two. Thus weak operational correspondence is much more flexible and allows to encode source term concepts that have no direct counterpart in the target.

Obtaining a result similar to Lemma 3.11 for weak operational correspondence is not that easy. Again this criterion is linked to a simulation condition on relations between source and target terms up to $\mathscr{R}_{\mathrm{T}}$, but weak operational correspondence does not directly map to a well-known kind of simulation relation. It is linked to a simulation relation that is in between coupled similarity and bisimilarity. We call it correspondence similarity.

Definition 3.12 (Correspondence Simulation) *A relation $\mathscr{R}$ is a* (weak reduction) correspondence simulation *if for each $(P,Q) \in \mathscr{R}$:*

- *$P \Longmapsto P'$ implies $\exists Q'.\ Q \Longmapsto Q' \wedge (P',Q') \in \mathscr{R}$*
- *$Q \Longmapsto Q'$ implies $\exists P'', Q''.\ P \Longmapsto P'' \wedge Q' \Longmapsto Q'' \wedge (P'',Q'') \in \mathscr{R}$*

Two terms are correspondence similar *if a correspondence simulation relates them.*

Just as coupled similarity, correspondence similarity allows for intermediate states that result from partial commitments, but in contrast to coupled similarity these intermediate states are not necessarily covered in the relation. Correspondence similarity is obviously strictly weaker than bisimilarity, but it implies coupled similarity.

Lemma 3.13 *For each correspondence simulation $\mathscr{R}$ there exists a coupled simulation $\mathscr{R}'$ such that $\forall (P,Q) \in \mathscr{R}.\ (P,Q),(Q,P) \in \mathscr{R}'$.*

Correspondence simulation is linked to weak operational correspondence in the same way as bisimilarity is linked to operational correspondence.

Lemma 3.14 (Weak Operational Correspondence) *$[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is weakly operat. corresp. w.r.t. a preorder $\mathscr{R}_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ that is a correspondence simulation iff $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S,[\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_{\mathrm{T}} = \mathscr{R}_{[\![\cdot]\!]}{\restriction_{\mathscr{P}_{\mathrm{T}}}}$ $\wedge\ \big(\forall S,T.\ (S,T) \in \mathscr{R}_{[\![\cdot]\!]} \to ([\![S]\!],T) \in \mathscr{R}_{\mathrm{T}}\big) \wedge \mathscr{R}_{[\![\cdot]\!]}$ is a preorder and a correspondence simulation.*

Accordingly, weak operational correspondence ensures that source terms and their literal translations are correspondence similar and thus coupled similar.

Correspondence similarity and coupled similarity are weaker than bisimilarity. Nevertheless, proving that a relation is a correspondence simulation and, even more, showing that a particular pair of terms is contained in a correspondence simulation, can be more difficult than it is in the case of bisimulation. Fortunately, encodings that satisfy only weak operational correspondence—and introduce partial commitments—often do so w.r.t. a variant of bisimilarity. As example consider the de-centralised encoding of [10]. It translates from CSP into asynchronous CCS with name passing and matching. [10] proves that this encoding is operational corresponding w.r.t. a target term preorder $\mathscr{R}_{\mathrm{T}}$ that is a weak reduction bisimulation. Thus, by Lemma 3.14, the encoding ensures that source terms and their literal translations are correspondence similar, and thus coupled similar.

4 Combining Encodability Criteria

As done in [8], often several different criteria are combined to ensure the quality of an encoding. Of course we have to ensure that the criteria we want to combine do not contradict each other and thus trivially rule out any kind of encoding. Moreover, the combination of criteria might lead to unexpected side effects, such that their combined effect on the quality of encodings is no longer obvious or clear. One major motivation of our desire to analyse encodability criteria is to be able to formally compare them and analyse side effects that result from their combinations.

In the previous section we derive iff-results linking a single criterion with the existence of a relation between source and target terms satisfying specific conditions. Of course we can trivially combine two

such results by considering two different source-target relations on the right-hand side. But this way side effects that results from the combination of the criteria remain hidden. Instead we want to combine the criteria into conditions of a single source-target relation. Therefore we need to find a witness, i.e., a relation that satisfies the conditions of both relations.

4.1 Divergence Reflection and Success Sensitiveness

The combinations of criteria defined on the pairs of $\mathscr{R}_{[\![\cdot]\!]}$—such as the preservation, reflection, or respect of some predicate—are easy to analyse. Obviously an encoding reflects divergence and respects success iff there exists a relation $\mathscr{R}_{[\![\cdot]\!]}$ that relates at least each source term to its literal translation and both reflects divergence and respects success.

Lemma 4.1 *Assume $\mathscr{L}_{\mathrm{S}}$ and $\mathscr{L}_{\mathrm{T}}$ each define a predicate $\downarrow_{\checkmark}\colon \mathscr{P} \to \mathbb{B}$. $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ reflects divergence and respects success iff $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S,[\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_{[\![\cdot]\!]}$ reflects divergence and respects success.*

The $\Leftarrow$-direction of Lemma 4.1 is an immediate corollary of Lemmas 3.3 and 3.6. For the other direction we obtain from these lemmata two relations that satisfy the condition $\mathsf{C} \triangleq \forall S.\ (S,[\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$ and of which one reflects divergence and the other respects success. We have to combine these two relations into a single relation that satisfies all three conditions. If the latter two conditions are defined on the pairs of the respective relations, this is always possible. The reason is that the condition C ensures that we can use $\{\,(S,[\![S]\!]) \mid S \in \mathscr{P}_{\mathrm{S}}\,\}$ as a witness for both relations and thus as a witness for their combined effect. More precisely, if there are two relations that both satisfy C and each satisfies a predicate about the pairs of the respective relation, then there exists a single relation, namely $\{\,(S,[\![S]\!]) \mid S \in \mathscr{P}_{\mathrm{S}}\,\}$, that satisfies all three conditions.

Lemma 4.2 *Let $\mathscr{R}_1, \mathscr{R}_2 \subseteq (\mathscr{P}_{\mathrm{S}} \uplus \mathscr{P}_{\mathrm{T}})^2$ and the predicates $\mathsf{P}_1, \mathsf{P}_2$ be such that $\forall i \in \{1,2\}\,.\,\forall S.\ (S,[\![S]\!]) \in \mathscr{R}_i$ and $\forall i \in \{1,2\}\,.\,\forall (P,Q) \in \mathscr{R}_i.\ \mathsf{P}_i((P,Q))$. Then there exists a relation $\mathscr{R}_{[\![\cdot]\!]} \subseteq (\mathscr{P}_{\mathrm{S}} \uplus \mathscr{P}_{\mathrm{T}})^2$ such that $\forall S.\ (S,[\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$ and $\forall i \in \{1,2\}\,.\,\forall (P,Q) \in \mathscr{R}_{[\![\cdot]\!]}.\ \mathsf{P}_i((P,Q))$.*

4.2 Adding Operational Correspondence

Gorla [8] combines five criteria to define 'good' encodings. Three of these—the 'semantical' ones—we considered in Section 3: weak operational correspondence (called 'operational correspondence' in [8]) w.r.t. a relation $\mathscr{R}_{\mathrm{T}}$, success sensitiveness, and divergence reflection. Gorla assumes that, 'for the sake of coherence' as he claims, the relation $\mathscr{R}_{\mathrm{T}}$ never relates two process T_P and T_Q such that $T_P \Downarrow_{\checkmark}$ and $T_Q \not\Downarrow_{\checkmark}$, i.e., $\mathscr{R}_{\mathrm{T}}$ has to respect (reachability of) success. This allows us to find a witness relation to combine the effect of weak operational correspondence and success sensitiveness. Our iff-result for weak operational correspondence requires that this relation is a preorder, has to relate source terms with their literal translations, and satisfies $\mathscr{R}_{\mathrm{T}} = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_{\mathrm{T}}}$. Because of that, a minimal witness is $\mathrm{t}(\mathrm{r}(\{\,(S,[\![S]\!]) \mid S \in \mathscr{P}_{\mathrm{S}}\,\} \cup \mathscr{R}_{\mathrm{T}}))$. This witness also satisfies $\forall S,T.\ (S,T) \in \mathscr{R}_{[\![\cdot]\!]} \to ([\![S]\!],T) \in \mathscr{R}_{\mathrm{T}}$. Without the condition that $\mathscr{R}_{\mathrm{T}}$ respects success—or another suitable assumption—we cannot ensure that $\mathrm{t}(\mathrm{r}(\{\,(S,[\![S]\!]) \mid S \in \mathscr{P}_{\mathrm{S}}\,\} \cup \mathscr{R}_{\mathrm{T}}))$ respects success and thus we find no witness for the combination of the respective conditions.

Lemma 4.3 *Assume $\mathscr{L}_{\mathrm{S}}, \mathscr{L}_{\mathrm{T}}$ each define a predicate $\downarrow_{\checkmark}\colon \mathscr{P} \to \mathbb{B}$. An encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is success sensitive and weakly operational corresponding w.r.t. a preorder $\mathscr{R}_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ that is a success respecting correspondence simulation iff $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S,[\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_{\mathrm{T}} = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_{\mathrm{T}}} \wedge\ \mathscr{R}_{[\![\cdot]\!]}$ respects success $\wedge$ $\big(\forall S,T.\ (S,T) \in \mathscr{R}_{[\![\cdot]\!]} \to ([\![S]\!],T) \in \mathscr{R}_{\mathrm{T}}\big) \wedge \mathscr{R}_{[\![\cdot]\!]}$ is a preorder and a correspondence simulation.*

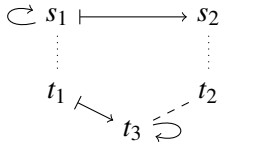


Figure 1: Encoding satisfying operational correspondence and divergence reflection

We obtain a similar result if we replace weak operational correspondence and correspondence simulation by operational correspondence and bisimulation. Similarly, we can replace weak operational correspondence, correspondence simulation, and the predicate $\Downarrow_{\checkmark}$ in the definition of success sensitiveness by strong operational correspondence, strong bisimulation, and $\downarrow_{\checkmark}$. Also, in all variants of the above result we can add on the left-hand side that the encoding as well as $\mathscr{R}_{\mathrm{T}}$ (weakly) respect barbs iff we add on the right-hand side that $\mathscr{R}_{[\![\cdot]\!]}$ (weakly) respects barbs.

As an example we can consider—once more—the de-centralised encoding from CSP into a variant of CCS of [10]. Additionally to operational correspondence w.r.t. a preorder $\mathscr{R}_{\mathrm{T}}$ that is a bisimulation, [10] proves that this encoding satisfies success sensitiveness, divergence reflection, barb sensitiveness—w.r.t. standard CSP barbs in the source and a notion of translated barbs in the target—and preservation of distributability (a criterion defined in [25]). $\mathscr{R}_{\mathrm{T}}$ respects success and weakly respects barbs (but does not reflect divergence). Thus the encoding ensures that source terms and their literal translations are correspondence similar and thus coupled similar w.r.t. a relation that respects success and weakly respects barbs.

Success sensitiveness significantly strengthens the requirements on a simulation relation like correspondence simulation or bisimulation. Thus the combined effect—a success respecting correspondence simulation—is stronger than the effects of both criteria considered in isolation—a correspondence simulation and a success respecting relation. Accordingly, the framework of Gorla in [8] ensures that (among other conditions) source terms and their literal translations are correspondence similar w.r.t. a success respecting relation and thus—to refer to a more established simulation relation—are coupled similar w.r.t. a success respecting relation.

In [8] there is no such condition that links $\mathscr{R}_{\mathrm{T}}$ and divergence reflection. Requiring that $\mathscr{R}_{\mathrm{T}}$ reflects divergence would e.g. exclude weak bisimulation. Since this relation is often referred to as *the* standard relation for calculi as the π-calculus, excluding it would be too strict a requirement. As a consequence, the criteria in [8] do not allow to combine the effects of weak operational correspondence and divergence reflection into a single relation, as done for success sensitiveness. Consider the following counterexample, visualised in Figure 1. Obviously the encoding—indicated by the dotted line—satisfies operational correspondence w.r.t. $\mathscr{R}_{\mathrm{T}}$—indicated by the dashed line—and reflects divergence. But to relate s_1 and its literal translation $[\![s_1]\!] = t_1$ by a correspondence simulation, we have to simulate the step $s_1 \longmapsto_{\mathrm{S}} s_2$. Therefore we need either the pair (s_2, t_3)—which can be obtained by including $\mathscr{R}_{\mathrm{T}}$ in $\mathscr{R}_{[\![\cdot]\!]}$—or the pair (s_2, t_1), but in either case the respective source-target relation does not reflect divergence. Thus in general an encoding that satisfies the criteria of [8] induces a source-target relation that is a correspondence simulation that only partially reflects divergence.

Of course particular encodings might satisfy stronger requirements than enforced by the minimal setting in [8]. If the encoding is operational corresponding w.r.t. a relation that reflects divergence, we can combine the effects of these two criteria in one relation. Accordingly, if an encoding reflects divergence, respects success, and satisfies operational correspondence w.r.t. a preorder that is a success respecting and divergence reflecting bisimulation, we can combine the conditions of all three relations

as in the following lemma.

Lemma 4.4 *Assume $\mathscr{L}_S$ and $\mathscr{L}_T$ each define a predicate $\downarrow_\checkmark$: $\mathscr{P} \to \mathbb{B}$. An encoding $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ reflects divergence, respects success, and is operational corresponding w.r.t. a preorder $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ that is a success respecting and divergence reflecting bisimulation iff $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S,[\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_T = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_T} \wedge \big(\forall S,T.\ (S,T) \in \mathscr{R}_{[\![\cdot]\!]} \to ([\![S]\!],T) \in \mathscr{R}_T\big) \wedge \mathscr{R}_{[\![\cdot]\!]}$ reflects divergence, respects success, and is a preorder and a bisimulation.*

Again we obtain similar results for weak operational correspondence and correspondence simulation as well as for strong operational correspondence, strong bisimulation, and $\downarrow_\checkmark$.

4.3 Full Abstraction and Operational Correspondence

Before the framework in [8] was proposed, often a combination of full abstraction and operational correspondence was used. For simplicity we switch to source-target relations that relate source terms and their literal translations in both directions and assume that $\mathscr{R}_S$ and $\mathscr{R}_T$ are equivalences in the following. Then a witness for the effect of operational correspondence is $\mathrm{t}(\mathrm{r}(\{\,(S,[\![S]\!]),([\![S]\!],S) \mid S \in \mathscr{P}_S\,\} \cup \mathscr{R}_T))$. Since this relation is transitive, it indeed suffices as witness to combine the effects of full abstraction and operational correspondence. The only obstacle left is that, to cover the effect of full abstraction, the source-target relation should also include $\mathscr{R}_S$. Fortunately we do not have to include $\mathscr{R}_S$ by construction, because its inclusion is ensured by full abstraction and the inclusion of $\mathscr{R}_T$. For every encoding $[\![\cdot]\!]$ that is fully abstract w.r.t. $\mathscr{R}_S$ and $\mathscr{R}_T$ and for all transitive relations $\mathscr{R}_{[\![\cdot]\!]}$ that relate at least all source terms to their literal translations in both directions, $\mathscr{R}_{[\![\cdot]\!]}$ contains $\mathscr{R}_S$ iff the restriction of $\mathscr{R}_{[\![\cdot]\!]}$ to encoded source terms contains the restriction of $\mathscr{R}_T$ to encoded sources.

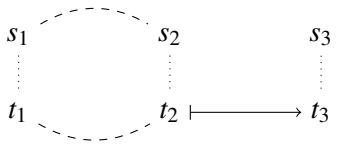
Lemma 4.5 *Let $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ be an encoding that is fully abstract w.r.t. $\mathscr{R}_S \subseteq \mathscr{P}_S^2$ and $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ and let $\mathscr{R}_{[\![\cdot]\!]} \subseteq (\mathscr{P}_S \uplus \mathscr{P}_T)^2$ be transitive such that $\forall S.\ (S,[\![S]\!]),([\![S]\!],S) \in \mathscr{R}_{[\![\cdot]\!]}$. Then $\mathscr{R}_S = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_S}$ iff $\forall S_1,S_2.\ ([\![S_1]\!],[\![S_2]\!]) \in \mathscr{R}_T \leftrightarrow ([\![S_1]\!],[\![S_2]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$.*

Because of that an encoding is fully abstract w.r.t. $\mathscr{R}_S$ and $\mathscr{R}_T$ and operational corresponding w.r.t. a bisimulation $\mathscr{R}_T$ iff there exists a transitive bisimulation that relates source terms and their literal translations in both directions and contains $\mathscr{R}_S$ and $\mathscr{R}_T$.

Lemma 4.6 *Let $\mathscr{R}_S \subseteq \mathscr{P}_S^2$ and $\mathscr{R}_T \subseteq \mathscr{P}_T^2$ be equivalences. An encoding $[\![\cdot]\!] : \mathscr{P}_S \to \mathscr{P}_T$ is fully abstract w.r.t. $\mathscr{R}_S$ and $\mathscr{R}_T$ and operational corresponding w.r.t. $\mathscr{R}_T$ and $\mathscr{R}_T$ is a bisimulation iff $\exists \mathscr{R}_{[\![\cdot]\!]}.$ $\big(\forall S.\ (S,[\![S]\!]),([\![S]\!],S) \in \mathscr{R}_{[\![\cdot]\!]}\big) \wedge \mathscr{R}_S = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_S} \wedge \mathscr{R}_T = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_T} \wedge \mathscr{R}_{[\![\cdot]\!]}$ is a transitive bisimulation.*

So what do we gain by combining the two criteria, that we do not obtain from each of them in isolation? In comparison to our iff-result for operational correspondence we add only the condition that $\mathscr{R}_S = \mathscr{R}_{[\![\cdot]\!]} \restriction_{\mathscr{P}_S}$. As a consequence $\mathscr{R}_S$ has to be a bisimulation.

Full abstraction ensures that $\mathscr{R}_S$ and $\mathscr{R}_T$ have the same basic properties. For example, if we either consider surjective encodings ($\forall T.\ \exists S.\ T = [\![S]\!]$) or restrict $\mathscr{R}_T$ to encoded source terms ($\{(T_1,T_2) \mid \exists S_1,S_2.\ T_1 = [\![S_1]\!] \wedge T_2 = [\![S_2]\!] \wedge (T_1,T_2) \in \mathscr{R}_T\}$), then $\mathscr{R}_S$ is reflexive iff $\mathscr{R}_T$ is reflexive, and similarly for symmetry and transitivity. But properties such as being a bisimulation are not respected by full abstraction on its own. As counterexample consider the fully abstract but not operational corresponding encoding in Figure 2. Here $\mathscr{R}_S$ is a bisimulation but $\mathscr{R}_T$ is not. By removing the arrow $t_2 \longmapsto_T t_3$ from the target and adding it to the source $s_2 \longmapsto_S s_3$, the encoding remains fully abstract and $\mathscr{R}_T$ becomes a bisimulation but $\mathscr{R}_S$ loses this property. Operational correspondence does not refer to a source relation $\mathscr{R}_S$ and thus does not enforce any properties on this relation. But combining full abstraction with operational correspondence w.r.t. a bisimulation $\mathscr{R}_T$ enforces $\mathscr{R}_S$ to be a bisimulation.



$\mathscr{L}_{\mathrm{S}} = (\{s_1, s_2, s_3\}, \emptyset)$
$\mathscr{L}_{\mathrm{T}} = (\{t_1, t_2, t_3\}, \{(t_2, t_3)\})$
$\mathscr{R}_{\mathrm{S}} = \mathrm{t}(\mathrm{s}(\mathrm{r}(\{(s_1, s_2)\})))$ and $\mathscr{R}_{\mathrm{T}} = \mathrm{t}(\mathrm{s}(\mathrm{r}(\{(t_1, t_2)\})))$
$[\![s_1]\!] = t_1$, $[\![s_2]\!] = t_2$, and $[\![s_3]\!] = t_3$

Figure 2: Encoding satisfying full abstraction but not operational correspondence

Lemma 4.7 *Let $\mathscr{R}_{\mathrm{S}} \subseteq \mathscr{P}_{\mathrm{S}}^2$ and $\mathscr{R}_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ be equivalences. If an encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is fully abstract w.r.t. $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$ and operational corresponding w.r.t. $\mathscr{R}_{\mathrm{T}}$ and $\mathscr{R}_{\mathrm{T}}$ is a bisimulation then $\mathscr{R}_{\mathrm{S}}$ is a bisimulation.*

To conclude that $\mathscr{R}_{\mathrm{S}}$ is a bisimulation iff $\mathscr{R}_{\mathrm{T}}$ is a bisimulation, we have to get rid of pairs in $\mathscr{R}_{\mathrm{T}}$ that do not result from pairs of encoded source terms and their derivatives, because operational correspondence provides no information about such pairs. The simplest way to do so, is to assume a surjective encoding.

Lemma 4.8 *Let $\mathscr{R}_{\mathrm{S}} \subseteq \mathscr{P}_{\mathrm{S}}^2$ and $\mathscr{R}_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ be equivalences. If an encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is surjective ($\forall T.\ \exists S.\ T = [\![S]\!]$), fully abstract w.r.t. $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$, and operational corresponding w.r.t. $\mathscr{R}_{\mathrm{T}}$ then:*

$\mathscr{R}_{\mathrm{S}}$ is a bisimulation iff $\mathscr{R}_{\mathrm{T}}$ is a bisimulation

5 Conclusions

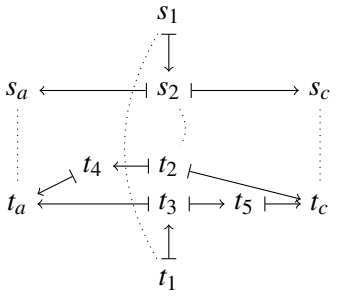
Within this paper we provide a number of results about different encodability criteria. In particular:

- We analyse divergence reflection, barb sensitiveness, success sensitiveness, full abstraction, and operational correspondence as well as several combinations of these criteria.
- We prove that different variants of operational correspondence correlate with different kinds of simulation relations from coupled similarity to strong bisimilarity.
- We define a new kind of simulation relation—correspondence similarity—that completely covers the effect of weak operational correspondence as proposed in [8].
- We relate the combination of success sensitiveness and operational correspondence w.r.t. a bisimulation with the existence of a success respecting bisimulation between source terms and their literal translations.
- We show that for surjective encodings the combination of full abstraction w.r.t. $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$ and operational correspondence w.r.t. $\mathscr{R}_{\mathrm{T}}$ implies that $\mathscr{R}_{\mathrm{S}}$ is a bisimulation iff $\mathscr{R}_{\mathrm{T}}$ is a bisimulation.

In [7] a quality criterion for encodings was proposed that requires the translation $[\![S]\!]$ of a source term S to be related to S according to a behavioural equivalence or preorder defined on a domain of interpretation (such as labelled transition systems or reduction-based transition systems with barbs) that applies to both languages. This behavioural relation has to be chosen with care and should be meaningful for the application at hand. Possible choices include strong and weak barbed bisimilarity, barbed weak coupled simulation equivalence, or (in between) our new correspondence preorder. Iff-results—as the results above—relate these instances of the criterion discussed in [7] with other encodability criteria. In particular, by the results of Section 3, if an encoding satisfies the criterion of [7] w.r.t. (weak) bisimilarity, then it also satisfies operational correspondence w.r.t. (weak) bisimilarity.[1] Moreover, if an encoding satisfies the criterion of [7] w.r.t. correspondence similarity, then it also satisfies weak operational correspondence w.r.t. coupled similarity.[2]

[1] And by the results of Section 4 the bisimulation may be required to (weakly) respect barbs at both sides of the implication.

[2] We may not conclude that it also satisfies weak operational correspondence w.r.t. correspondence similarity, at least not when also weakly respecting barbs. A counterexample can be found in Figure 3. There we have a weakly barb respecting



$\mathscr{L}_{\mathrm{S}} = (\{s_1, s_2, s_a, s_c\}, \{(s_1, s_2), (s_2, s_a), (s_2, s_c)\})$
$s_a \downarrow_{\mathscr{L}_{\mathrm{S}}} a \qquad s_c \downarrow_{\mathscr{L}_{\mathrm{S}}} c \qquad s_2 \downarrow_{\mathscr{L}_{\mathrm{S}}} b$
$\mathscr{L}_{\mathrm{T}} = (\{t_1, t_2, t_3, t_4, t_5, t_a, t_c\},$
$\{(t_1, t_3), (t_2, t_4), (t_2, t_c), (t_3, t_5), (t_3, t_a), (t_4, t_a), (t_5, t_c)\})$
$t_a \downarrow_{\mathscr{L}_{\mathrm{T}}} a \qquad t_c \downarrow_{\mathscr{L}_{\mathrm{T}}} c \qquad t_4 \downarrow_{\mathscr{L}_{\mathrm{T}}} b \qquad t_5 \downarrow_{\mathscr{L}_{\mathrm{T}}} b$
$\mathscr{R}_{\textit{corr. sim.}} = \{(s_1, t_1), (s_2, t_2), (s_2, t_3), (s_a, t_a), (s_c, t_c)\}$
$[\![s_1]\!] = t_1$, $[\![s_2]\!] = t_2$, $[\![s_a]\!] = t_a$, and $[\![s_c]\!] = t_c$

Figure 3: Encoding not satisfying weak operational correspondence w.r.t. to a correspondence simulation that weakly respects barbs, even though $\mathscr{R}_{\textit{corr. sim.}}$ is a weakly barb respecting correspondence simulation relating each source term with its encoding

The above results may leave the impression that we try to replace common encodability criteria by conditions on relations between source terms and their translations. That is not the case. But we provide alternative ways to prove different criteria. An example of how the above results can be used to reason about the quality of an encoding are the two encodings of [10]. That paper analyses ways to encode the CSP synchronisation mechanism following an approach of [21]. The latter shows that a central encoding of a similar synchronisation mechanism ensures that source terms and their translations behave bisimilar, whereas a decentralised encoding only ensures coupled similarity. Proving coupled similarity can be more difficult than proving bisimilarity. Here our results allow to decrease the proof burden. With Lemma 3.14 we can conclude from weak operational correspondence w.r.t. a bisimulation that source terms and their literal translations are coupled similar, without having to deal with coupled similarity directly. This way we have to deal with the difficult partial commitments, which are introduced by the de-central implementation, only in operational correspondence and not when relating target terms.

In retrospective, mapping encodability criteria on requirements of a relation between source and target terms seems quite natural. Indeed the main challenge of the above presented iff-results was not in proving them but in finding the exact matches between variants of the considered criteria and requirements on the relation. As a consequence we had to define a new kind of simulation relation to capture the version of operational correspondence used in [8].

We do not claim that it is always simple to obtain iff-results as presented in this paper or that we provide a strategy to obtain such results. Instead we claim that proving such results formally captures the effect of a criterion on the quality of an encoding function and thus(1) helps us to understand a criterion, (2) allows to identify unexpectedly strict or weak criteria (3) allows to compare (sets of) criteria, and (4) allows to analyse the side effects that result from the combination of criteria. Analysing criteria this way is not necessarily straightforward. To illustrate this, consider the requirement on the preservation of the (degree of) distribution of a process (preservation of distributability). In the context of asynchronous distributed systems this requirement is very important.

Several attempts to capture it were proposed in the literature. At least for the π-calculus, the most

correspondence simulation relating s_2 with t_2 and s_2 with t_3, but, due to the asymmetric nature of correspondence simulations, there is no weakly barb respecting correspondence simulation relating t_2 and t_3.[3] By Lemma 3.13 the terms t_2 and t_3 are weakly barb respecting coupled similar, however.

[3]This example does not contradict the weakly barbed variant of Lemma 4.3, for its right-hand side does not hold. Namely, the condition $\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$ forces $(s_1, t_1) \in \mathscr{R}_{[\![\cdot]\!]}$, and thus, since $\mathscr{R}_{[\![\cdot]\!]}$ is a correspondence simulation, also $(s_2, t) \in \mathscr{R}_{[\![\cdot]\!]}$ for some $t \in \{t_1, t_3, t_5, t_a, t_c\}$. As s_2 weakly respects barbs a and c, so must t, yielding $t \in \{t_1, t_3\}$. The requirement $(S, T) \in \mathscr{R}_{[\![\cdot]\!]} \rightarrow ([\![S]\!], T) \in \mathscr{R}_{\mathrm{T}}$ yields $(t_2, t) \in \mathscr{R}_{[\![\cdot]\!]}$, but there exists no correspondence simulation containing this pair.

prominent candidate is the homomorphic translation of the parallel operator ($[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$) as used in [17]. [25] shows that this criterion is too strict by providing an encoding that preserves distributability but does not translate the parallel operator homomorphically. Instead this encoding is compositional. Compositionality requires that the translation of an operator is the same for all occurrences of that operator in a term, i.e., it can be captured by a context. Compositionality is significantly weaker than the homomorphic translation of the parallel operator but also forbids the introduction of global coordinators. However, to ensure the preservation of distributability, this criterion is too weak. [25] claims to provide a suitable criterion for the degree of distributability, but without a formal way to reason about the effect of encodability criteria, there is no way to formally prove such a claim. Thus [25] can only provide arguments and illustrations. The inability to formally prove it was one of the original motivations for the present work. Unfortunately, analysing the three criteria compositionality, the homomorphic translation of the parallel operator, and the preservation of distributability is not an easy task.

Compositionality obviously implies some kind of congruence property on encoded source term contexts, but it is not obvious how to turn this observation into an iff-result. To map the homomorphic translation of the parallel operator on conditions on a relation between source and target terms, is even more difficult. This criterion clearly implies some strong properties on such a relation, but it is not clear which condition implies the homomorphic translation of the parallel operator. Because of that, we cannot completely capture the effect of this criterion on the quality of an encoding. This explains why this criterion was originally accepted as a criterion for the preservation of distributability. It is not easy to capture the cases for which it is too strict. The criterion for the preservation of distributability proposed in [25] can intuitively be understood as a concurrency respecting variant of operational correspondence. It not only requires that the source term behaviour is preserved and reflected, but also that the simulations of independent steps are independent. Thus analysing this criterion appears to require some kind of simulation relations that not only consider interleaving semantics. We leave the analysis of these criteria to further research.

All claims in this paper have been proved using the interactive theorem prover Isabelle/HOL. For this purpose, a rich theory of encodability criteria was implemented. Since we do not force any assumptions on process calculi except that they consist of a set of processes, i.e., a type $\mathscr{P}$, and a reduction relation, i.e., a relation of type $\mathscr{P}^2 = \mathscr{P} \times \mathscr{P}$, this theory can be used to formally reason in Isabelle about encodings for all kinds of source and target languages. A number of well-known process calculi including the π-calculus can for instance be represented in the Psi-calculi framework [1]. Thus there are Isabelle implementations of well-known process calculi that can directly be combined with our Isabelle implementation to formally reason about encodings between such calculi.

References

[1] J. Bengtson, M. Johansson, J. Parrow & B. Victor (2009): *Psi-calculi: Mobile Processes, Nominal Data, and Logic*. In: *Proceedings of LICS*, IEEE, pp. 39–48, doi:10.1109/LICS.2009.20.

[2] N. Busi, M. Gabbrielli & G. Zavattaro (2009): *On the expressive power of recursion, replication and iteration in process calculi*. *Mathematical Structures in Computer Science* 19(6), pp. 1191–1222, doi:10.1017/S096012950999017X.

[3] D. Cacciagrano, F. Corradini, J. Aranda & F.D. Valencia (2008): *Linearity, Persistence and Testing Semantics in the Asynchronous Pi-Calculus*. *Electronic Notes in Theoretical Computer Science* 194(2), pp. 59–84, doi:10.1016/j.entcs.2007.11.006.

[4] L. Cardelli & A. D. Gordon (2000): *Mobile ambients*. *Theoretical Computer Science* 240(1), pp. 177–213, doi:10.1016/S0304-3975(99)00231-5.

[5] R. J. van Glabbeek (2001): *The Linear Time – Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes*. Handbook of Process Algebra, pp. 3–99, doi:10.1016/B978-044482830-9/50019-9.

[6] R. J. van Glabbeek (1994): *On the expressiveness of ACP (extended abstract)*. In: Proceedings of ACP, Workshops in Computing, Springer, pp. 188–217, doi:10.1007/978-1-4471-2120-6_8.

[7] R.J. van Glabbeek (2012): *Musings on Encodings and Expressiveness*. In: Proceedings of EXPRESS/SOS, EPTCS 89, pp. 81–98, doi:10.4204/EPTCS.89.7.

[8] D. Gorla (2010): *Towards a unified approach to encodability and separation results for process calculi*. Information and Computation 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[9] D. Gorla & U. Nestmann (2014): *Full abstraction for expressiveness: history, myths and facts*. Mathematical Structures in Computer Science, pp. 1–16, doi:10.1017/S0960129514000279.

[10] M. Hatzel, C. Wagner, K. Peters & U. Nestmann (2015): *Encoding CSP into CCS*. In: Proceedings of EXPRESS/SOS, EPTCS 190, pp. 61–75, doi:10.4204/EPTCS.190.5.

[11] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes, Part I and II*. Information and Computation 100(1), pp. 1–77, doi:10.1016/0890-5401(92)90008-4.

[12] J. C. Mitchell (1993): *On abstraction and the expressive power of programming languages*. Science of Computer Programming 21(2), pp. 141–163, doi:10.1016/0167-6423(93)90004-9.

[13] U. Nestmann (2000): *What is a "Good" Encoding of Guarded Choice?* Information and Computation 156(1-2), pp. 287–319, doi:10.1006/inco.1999.2822.

[14] U. Nestmann (2006): *Welcome to the Jungle: A Subjective Guide to Mobile Process Calculi*. In: Proceedings of CONCUR, LNCS 4137, Springer, pp. 52–63, doi:10.1007/11817949_4.

[15] U. Nestmann & B. C. Pierce (2000): *Decoding Choice Encodings*. Information and Computation 163(1), pp. 1–59, doi:10.1006/inco.2000.2868.

[16] T. Nipkow, L. C. Paulson & M. Wenzel (2002): *Isabelle/HOL: a proof assistant for higher-order logic*. 2283, Springer Science & Business Media, doi:10.1007/3-540-45949-9_1.

[17] C. Palamidessi (2003): *Comparing The Expressive Power Of The Synchronous And Asynchronous Pi-Calculi*. Mathematical Structures in Computer Science 13(5), pp. 685–719, doi:10.1017/S0960129503004043.

[18] C. Palamidessi, V. A. Saraswat, F. D. Valencia & B. Victor (2006): *On the Expressiveness of Linearity vs Persistence in the Asychronous Pi-Calculus*. In: Proceedings of LICS, IEEE Computer Society, pp. 59–68, doi:10.1109/LICS.2006.39.

[19] J. Parrow (2008): *Expressiveness of Process Algebras*. Electronic Notes Theoretical Computer Science 209, pp. 173–186, doi:10.1016/j.entcs.2008.04.011.

[20] J. Parrow (2014): *General conditions for full abstraction*. Mathematical Structures in Computer Science, pp. 1–3, doi:10.1017/S0960129514000280.

[21] J. Parrow & P. Sjödin (1992): *Multiway synchronization verified with coupled simulation*. In: Proceedings of CONCUR, LNCS 630, Springer, pp. 518–533, doi:10.1007/BFb0084813.

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

[23] K. Peters (2012): *Translational Expressiveness*. Ph.D. thesis, TU Berlin. Available at `http://opus.kobv.de/tuberlin/volltexte/2012/3749/`.

[24] K. Peters & U. Nestmann (2012): *Is it a "Good" Encoding of Mixed Choice?* In: Proceedings of FOSSACS, LNCS 7213, Springer, pp. 210–224, doi:10.1007/978-3-642-28729-9_14.

[25] K. Peters, U. Nestmann & U. Goltz (2013): *On Distributability in Process Calculi*. In: Proceedings of ESOP, LNCS 7792, Springer, pp. 310–329, doi:10.1007/978-3-642-37036-6_18.

[26] J. G. Riecke (1991): *Fully abstract translations between functional languages*. In: Proceedings of POPL, ACM, pp. 245–254, doi:10.1145/99583.99617.

Encoding CSP into CCS *

Meike Hatzel
TU Berlin

Christoph Wagner
TU Berlin

Kirstin Peters†
TU Dresden

Uwe Nestmann
TU Berlin

We study encodings from CSP into asynchronous CCS with name passing and matching, so in fact, the asynchronous π-calculus. By doing so, we discuss two different ways to map the multi-way synchronisation mechanism of CSP into the two-way synchronisation mechanism of CCS. Both encodings satisfy the criteria of Gorla except for compositionality, as both use an additional top-level context. Following the work of Parrow and Sjödin, the first encoding uses a centralised coordinator and establishes a variant of weak bisimilarity between source terms and their translations. The second encoding is decentralised, and thus more efficient, but ensures only a form of coupled similarity between source terms and their translations.

1 Introduction

In the context of a scientific meeting on Expressiveness in Concurrency and Structural Operational Semantics (SOS), likely very little needs to be said about the process algebras (or process calculi) CSP and CCS. Too many papers have been written since their advent in the 70's to be mentioned in our own paper; it is instructive, though, and recommended to appreciate Jos Baeten's historical overview [1], which also places CSP and CCS in the context of other process algebras like ACP and the many extensions by probabilities, time, mobility, etc. Here, we just select references that help to understand our motivation.

Differences. From the beginning, although CSP [8] and CCS [11] were intended to capture, describe and analyse reactive and interactive concurrent systems, they were designed following rather different philosophies. Tony Hoare described this nicely in his position paper [9] as follows: "A primary goal in the original design of CCS was to discover and codify a minimal set of basic primitive agents and operators ... and a wide range of useful operators which have been studied subsequently are all definable in terms of CCS primitives." and "CSP was more interested in this broader range of useful operators, independent of which of them might be selected as primitive." So, at their heart, the two calculi use two different synchronisation mechanisms, one (CCS) using binary, i.e., two-way, handshake via matching actions and co-actions, the other (CSP) using multiway synchronisation governed by explicit synchronisation sets that are typically attached to parallel composition. Another difference is the focus on Structural Operational Semantics in CCS, and the definition of behavioural equivalences on top of this, while CSP emphasised a trace-based denotational model, enhanced with failures, and the question on how to design models such that they satisfy a given set of laws of equivalence.

Comparisons. From the early days, researchers were interested in more or less formal comparisons between CSP and CCS. This was carried out by both Hoare [9] and Milner [12] themselves, where they concentrate on the differences in the underlying design principles. But also other researchers joined the game, but with different analysis tools and comparison criteria.

For example, Brookes [3] contributed a deep study on the relation between the underlying abstract models, synchronisation trees for CCS and the failures model of CSP. Quite differently, Lanese and Montanari [10] used the power to transform graphs as a measure for the expressiveness of the two calculi.

*Supported by the DFG via project "Synchronous and Asynchronous Interaction in Distributed Systems".
†Supported by the German Federal and State Governments via the Excellence Initiative (Institutional Strategy).

S. Crafa and D. Gebler (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2015)
EPTCS 190, 2015, pp. 61–75, doi:10.4204/EPTCS.190.5

© Hatzel, Wagner, Peters & Nestmann
This work is licensed under the Creative Commons Attribution License.

Yet completely differently, Parrow and Sjödin [16, 21] tried to find an algorithm to implement—best in a fully distributed fashion—the multiway synchronisation operator of CSP (and its variant LOTOS [2]) using the supposedly simpler two-way synchronisation of CCS. They came up with two candidates—a reasonably simple centralised synchroniser, and a considerably less simple distributed synchroniser[1]—and proved that the two are not weakly bisimilar, but rather coupled similar, which is only slightly weaker. Coupled simulation is a notion that Parrow and Sjödin invented for just this purpose, but it has proved afterwards to be often just the right tool when analysing the correctness of distribution- and divergence-sensitive encodings that involve partial commitments (whose only effect is to gradually perform internal choices) [15].

The probably most recent comparison between CSP and CCS was provided by van Glabbeek [5]. As an example for his general framework to analyse the relative expressive power of calculi, he studied the existence of syntactical translations from CSP into CCS, for which a common semantical domain is provided via labelled transition systems (LTS) derived from respective sets of SOS rules. The comparison is here carried out by checking whether a CSP term and its translation into CCS are distinguishable with respect to a number of equivalences defined on top of the LTS. The concrete results are: (1) there is a translation that is correct up to trace equivalence (and contains deadlocks), and (2) there is no translation that is correct up to weak bisimilarity equivalence that also takes divergence into account.

Contribution. Given van Glabbeek's negative result, and given Parrow and Sjödin's algorithm, we set out to check whether we can define a syntactical encoding from CSP into CCS—using Parrow and Sjödin's ideas—that is correct up to coupled similarity.[2] We almost managed. In this paper, we report on our current results along these lines: (1) Our encoding target is an asynchronous variant of CCS, but enhanced with name-passing and matching, so it is in fact an asynchronous π-calculus; we kept mentioning CCS in the title of this paper, as it clearly emphasises the origin and motivation of this work. But, we could *not* do without name-passing. (2) We exhibit one encoding that is not distributability-preserving (so, it represents a centralised solution), but is correct up to weak bisimilarity and does not introduce divergence. This does not contradict van Glabbeek's results, but suggests that van Glabbeek's framework implies some form of distributability-preservation. (3) We exhibit another encoding that *is* distributability-preserving and divergence-reflecting, but is only correct up to coupled similarity.

Overview. We introduce the considered variants of CSP and CCS in § 2. There we also introduce the criteria—that are (variants of) the criteria in [6] and [20]—modulo which we prove the quality of the considered encodings. In § 3 we introduce the inner layer of our two encodings. It provides the main machinery to encode synchronisations of CSP. We complete this encoding with an outer layer that is either a centralised (§ 4) or a decentralised coordinator (§ 5). In § 6 we discuss the two encodings. Missing proofs and some additional informations can be found in [7].

2 Technical Preliminaries

A process calculus $(\mathscr{P}, \longmapsto)$ consists of a set $\mathscr{P}$ of processes (syntax) and a reduction relation $\longmapsto \subseteq \mathscr{P}^2$ (semantics). Let $\mathscr{N}$ be the countably-infinite set of names. $\tau \notin \mathscr{N}$ denotes an internal unobservable action. We use $a, b, x, \ldots$ to range over names and $P, Q, \ldots$ to range over processes. We use $\alpha, \beta \ldots$ to range over $\mathscr{N} \cup \{\tau\}$. $\tilde{a}$ denotes a sequence of names. Let $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$ denote the sets of free names and bound names occurring in P, respectively. Their definitions are completely standard. We use $\sigma, \sigma', \sigma_1, \ldots$

[1] Recently [4], a slight variant of the protocol behind this algorithm was used to implement the distributed compiler DLC for a substantial subset of LNT (successor of LOTOS New Technology) that yields reasonably efficient C code.

[2] The idea and a first draft of the encoding were developed by Nestmann and van Glabbeek during a stay at NICTA, Sydney.

to range over substitutions. A substitution is a mapping $[{}^{x_1}/_{y_1},\ldots,{}^{x_n}/_{y_n}]$ from names to names. The application of a substitution on a term $P[{}^{x_1}/_{y_1},\ldots,{}^{x_n}/_{y_n}]$ is defined as the result of simultaneously replacing all free occurrences of y_i by x_i for $i \in \{1,\ldots,n\}$. For all names in $\mathscr{N} \setminus \{y_1,\ldots,y_n\}$ the substitution behaves as the identity mapping. The relation $\longmapsto$ as defined in the semantics below defines the reduction steps processes can perform. We write $P \longmapsto P'$ if $(P,P') \in \longmapsto$ and call P' a *derivative* of P. Let $\Longmapsto$ denote the reflexive and transitive closure of $\longmapsto$. P is *divergent* if it has an infinite sequence of steps $P \longmapsto^{\omega}$. We use *barbs* or *observables* to distinguish between processes with different behaviours. We write $P{\downarrow_\alpha}$ if P has a barb α, where the predicate $\cdot\downarrow_\cdot$ can be defined differently for each calculus. Moreover P has a weak barb α, if P may reach a process with this barb, i.e., $P{\Downarrow_\alpha} \triangleq \exists P'.\, P \Longmapsto P' \wedge P'{\downarrow_\alpha}$.

As source calculus we use the following variant of CSP [8].

Definition 1. The processes $\mathscr{P}_{\mathrm{CSP}}$ are given by

$$P ::= P\|_A P \quad | \quad \mathrm{DIV} \quad | \quad \mathrm{STOP} \quad | \quad P \sqcap P \quad | \quad P/b \quad | \quad f(P) \quad | \quad X \quad | \quad \mu X \cdot P \quad | \quad \Sigma_{i\in\mathscr{I}} a_i \to P$$

where $X \in \mathscr{X}$ is a process variable, $A \subseteq \mathscr{N}$, and $\mathscr{I}$ is a finite index set.

$P\|_A Q$ is the parallel composition of P and Q, where P and Q can proceed independently except for actions $a \in A$, on which they have to synchronise. DIV describes *divergence*. STOP denotes *inaction*. *Internal choice* $P \sqcap Q$ reduces to either P or Q within a single internal step. *Concealment* P/b hides an action b and masks it as τ. *Renaming* $f(P)$ for some $f : \mathscr{N} \to \mathscr{N}$ extended by $f(\tau) = \tau$ behaves as P, where a is replaced by $f(a)$ for all $a \in \mathscr{N}$. *Recursion* $\mu X \cdot P$ describes a process behaving like P with every occurrence of X being replaced by $\mu X \cdot P$. *External choice* $\Sigma_{i\in\mathscr{I}} a_i \to P_i$ offers a selection of one of the *action prefixes* $a_i \to \cdot$ followed by the corresponding continuation P_i, so it may perform any a_i and then behave like P_i. Note that we enforce action prefixes to be syntactically part of an external choice construct. As usual, we use $M \,\Box\, N$ to denote binary external choice.

The CSP semantics is given by the following rules, using labelled steps $\xrightarrow{\alpha}$ to define $\longmapsto$:

$$\frac{E \xrightarrow{b} E'}{E/b \xrightarrow{\tau} E'/b} \qquad \frac{E \xrightarrow{\alpha} E' \quad (\alpha \neq b)}{E/b \xrightarrow{\alpha} E'/b} \qquad \frac{E \xrightarrow{\alpha} E'}{f(E) \xrightarrow{f(\alpha)} f(E')} \qquad \frac{M_j \xrightarrow{a} M_j' \quad (j \in \mathscr{I})}{\Sigma_{i\in\mathscr{I}} M_i \xrightarrow{a} M_j'}$$

$$(a \to E) \xrightarrow{a} E \qquad \mu X \cdot E \xrightarrow{\tau} E[\mu X \cdot E/X]$$

$$\frac{E \xrightarrow{\alpha} E' \quad (\alpha \notin A)}{E\|_A F \xrightarrow{\alpha} E'\|_A F} \qquad \frac{F \xrightarrow{\alpha} F' \quad (\alpha \notin A)}{E\|_A F \xrightarrow{\alpha} E\|_A F'} \qquad \frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F' \quad (a \in A)}{E\|_A F \xrightarrow{a} E'\|_A F'} \qquad \frac{P \xrightarrow{\tau} P'}{P \longmapsto P'}$$

$$\mathrm{DIV} \xrightarrow{\tau} \mathrm{DIV} \qquad E \sqcap F \xrightarrow{\tau} E \qquad E \sqcap F \xrightarrow{\tau} F$$

A barb of CSP is the possibility of a term, to perform an action, i.e., $P{\downarrow_a} \triangleq \exists P'.\, P \xrightarrow{a} P'$. Following the definition of distributability in [20] a CSP term P is distributable into $P_1,\ldots,P_n$ if $P_1,\ldots,P_n$ are unguarded subterms of P such that every action prefix in P occurs in exactly one of the $P_1,\ldots,P_n$, where different but equally-named action prefixes are distinguished and unguarded occurrences of $\mu X \cdot P'$ may result in several copies of P' within the $P_1,\ldots,P_n$.

As target calculus we use an asynchronous variant of CCS [11] with name-passing and matching.

Definition 2. The processes $\mathscr{P}_{\mathrm{CCS}}$ are given by

$$P ::= P \mid P \quad | \quad (\nu \tilde{c})P \quad | \quad *\underline{c}(\tilde{x}).P \quad | \quad \underline{c}(\tilde{x}).P \quad | \quad \overline{c}\langle\tilde{x}\rangle \quad | \quad [c=z]P \quad | \quad \mathbf{0}$$

$P \mid Q$ is the parallel composition of P and Q, where P and Q can either proceed independently or synchronise on matching channels names. $(\nu\tilde{c})\,P$ restricts the visibility of actions using names in $\tilde{c}$ to P. $\underline{c}(\tilde{x}).P$ denotes input on channel c. $\overline{c}\langle\tilde{x}\rangle$ is output on channel c. Since there is no continuation, we interpret this calculus as asynchronous. We use $*\underline{c}(\tilde{x}).P$ to denote *replicated input* on channel c with the continuation P. $[x=y]\,P$ is the matching operator, if $x=y$ then P is enabled. $\mathbf{0}$ denotes inaction.

The CCS semantics is given by following transition rules:

$$\frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad \frac{P \longmapsto P'}{(\nu\tilde{c})\,P \longmapsto (\nu\tilde{c})\,P'} \qquad \frac{P \equiv P' \quad P' \longmapsto Q' \quad Q' \equiv Q}{P \longmapsto Q}$$

$$*\underline{c}(\tilde{x}).P \mid \overline{c}\langle\tilde{y}\rangle \longmapsto *\underline{c}(\tilde{x}).P \mid P[\tilde{y}/\tilde{x}] \qquad \overline{c}\langle\tilde{y}\rangle \mid \underline{c}(\tilde{x}).Q \longmapsto P \mid Q[\tilde{y}/\tilde{x}]$$

where $\equiv$ denotes structural congruence given by the rules: $P \mid 0 \equiv P$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$, $(\nu\tilde{a})\,\mathbf{0} \equiv \mathbf{0}$, $P \mid (\nu\tilde{a})\,Q \equiv (\nu\tilde{a})(P \mid Q)$ if $\mathrm{bn}(\tilde{a}) \notin \mathrm{fn}(P)$, and $[x=x]\,P \equiv P$. As discussed in [20], a CCS term P is distributable into $P_1,\ldots,P_n$ if $P \equiv (\nu\tilde{x})(P_1 \mid \ldots \mid P_n)$.

Simulation Relations. The semantics of a process is usually considered modulo some behavioural equivalence. For many calculi, *the* standard reference equivalence is some form of weak bisimilarity. In the context of encodings, the source and target language often differ in their relevant obervables, i.e., barbs. In this case, it is advantageous to use a variant of reduction bisimilarity. With Gorla [6], we add a *success* operator $\checkmark$ to the syntax of both CSP and CCS. Since $\checkmark$ cannot be further reduced, the semantics is left unchanged in both cases. The test for the reachability of success is standard in both languages, i.e., $P\downarrow_{\checkmark} \triangleq \exists P'.\, P \equiv \checkmark \mid P'$. To obtain a non-trivial equivalence, we require that the bisimulation respects success and the reachability of barbs. We use the standard definition of barbs in CSP, i.e., action prefixes. Our encoding function will translate all source terms into closed terms, thus the standard definition of CCS barbs would not provide any information. Instead we use a notion of translated barb ($\cdot\Downarrow_{[\![\cdot]\!]}$) that reflects how the encoding function translates source term barbs. Its definition is given in Section 3.

Definition 3 (Bisimulation)**.** A relation $\mathscr{R} \subseteq \mathscr{P}^2$ is a *(success-sensitive, [translated-]barb-respecting, weak, reduction) bisimulation* if, whenever $(P,Q) \in \mathscr{R}$, then:

- $P \longmapsto P'$ implies $\exists Q'.\, Q \Longmapsto Q' \wedge (P',Q') \in \mathscr{R}$
- $Q \longmapsto Q'$ implies $\exists P'.\, P \Longmapsto P' \wedge (P',Q') \in \mathscr{R}$
- $P\Downarrow_{\checkmark}$ iff $Q\Downarrow_{\checkmark}$
- P and Q reach the same (translated) barbs, where we use $\cdot\Downarrow_a$ for CSP and $\cdot\Downarrow_{[\![a]\!]}$ for CCS

Two terms $P,Q \in \mathscr{P}$ are *bisimilar*, denoted as $P \mathrel{\dot{\approx}} Q$, if there exists a bisimulation that relates P and Q.

We use the symbol $\dot{\approx}$ to denote either bisimilarity on our target language CCS or on the disjoint union of CSP and CCS that allows us to describe the relationship between source terms and their translations. In the same way we define a corresponding variant of coupled similarity.

Definition 4 (Coupled Simulation)**.** A relation $\mathscr{R} \subseteq \mathscr{P}^2$ is a *(success-sensitive, [translated-]barb-respecting, weak, reduction) coupled simulation* if, whenever $(P,Q) \in \mathscr{R}$, then:

- $P \longmapsto P'$ implies $\exists Q'.\, Q \Longmapsto Q' \wedge (P',Q') \in \mathscr{R}$ and $\exists Q''.\, Q \Longmapsto Q'' \wedge (Q'',P') \in \mathscr{R}$
- $P\Downarrow_{\checkmark}$ iff $Q\Downarrow_{\checkmark}$
- P and Q reach the same (translated) barbs, where we use $\cdot\Downarrow_a$ for CSP and $\cdot\Downarrow_{[\![a]\!]}$ for CCS

Two terms $P,Q \in \mathscr{P}$ are *coupled similar*, denoted as $P \mathrel{\dot{\approx}}_{\mathrm{cs}} Q$, if there exists a coupled simulation that relates P and Q in both directions.

Encodings and Quality Criteria. We consider two different translations from (the above-defined variant of) CSP into (the above-defined variant of) CCS with name passing and matching. In this context, we

refer to CSP terms as *source terms* $\mathscr{P}_{\mathrm{S}}$ and to CCS terms as *target terms* $\mathscr{P}_{\mathrm{T}}$. Encodings often translate single source steps into a sequence or pomset of target steps. We call such a sequence or pomset a *simulation* of the corresponding source term step. Moreover, we assume for each encoding the existence of a so-called renaming policy φ, i.e., a mapping of names from the source into vectors of target term names.

To analyse the quality of encodings and to rule out trivial or meaningless encodings, Gorla [6] provide a general framework comprising five quality criteria, which have afterwards been used in many papers. In addition to our above-mentioned definition of process calculus, whough, Gorla requires the target calculus to be equipped with a notion of behavioural equivalence $\asymp$ on target terms. Its purpose is to describe the 'abstract' behaviour of a target process, where 'abstract' refers to an observer at the source level. In [6], the equivalence $\asymp$ is often defined as a barbed equivalence (cf. [13]) or can be derived directly from the reduction semantics, and it typically is a congruence, at least with respect to parallel composition. Bisimilarity and coupled similarity are such relations on CCS terms. The criteria are:

(1) *Compositionality*: The translation of an operator op is the same for all occurrences of that operator in a term, i.e., it can be captured by a context $\mathscr{C}_{\mathrm{op}}$ such that $\mathrm{enc}(\mathrm{op}(x_1,\ldots,x_n,S_1,\ldots,S_m)) = \mathscr{C}^N_{\mathbf{op}}(x_1,\ldots,x_n,\mathrm{enc}(S_1),\ldots,\mathrm{enc}(S_m))$ for $\mathrm{fn}(S_1)\cup\ldots\cup\mathrm{fn}(S_m) = N$.

(2) *Name Invariance*: The encoding does not depend on particular names, i.e., for every S and σ, it holds that $\mathrm{enc}(\sigma(S)) \equiv \sigma'(\mathrm{enc}(S))$ if σ is injective and $\mathrm{enc}(\sigma(S)) \asymp \sigma'(\mathrm{enc}(S))$ otherwise, where σ' is such that $\varphi(\sigma(n)) = \sigma'(\varphi(n))$ for every $n \in \mathscr{N}$.

(3) *Operational Correspondence*: Every computation of a source term can be simulated by its translation, i.e., $S \Longmapsto_{\mathrm{S}} S'$ implies $\mathrm{enc}(S) \Longmapsto_{\mathrm{T}} \asymp \mathrm{enc}(S')$ (completeness), and every computation of a target term corresponds to some computation of the corresponding source term (soundness, compare to Section 5).

(4) *Divergence Reflection*: The encoding does not introduce divergence, i.e., $\mathrm{enc}(S) \longmapsto^{\omega}_{\mathrm{T}}$ implies $S \longmapsto^{\omega}_{\mathrm{S}}$.

(5) *Success Sensitiveness*: A source term and its encoding answer the tests for success in exactly the same way, i.e., $S\Downarrow_{\checkmark}$ iff $\mathrm{enc}(S)\Downarrow_{\checkmark}$.

Our encodings will satisfy all of these criteria except for compositionality, because both encodings consists of two layers. [20] shows that the above criteria do not ensure that an encoding preserves distribution and proposes an additional criterion for the preservation of distributability.

Definition 5 (Preservation of Distributability)**.** An encoding $\mathrm{enc}(\cdot)$ *preserves distributability* if for every S and for all terms $S_1,\ldots,S_n$ that are distributable within S there are some $T_1,\ldots,T_n$ that are distributable within $\mathrm{enc}(S)$ such that $T_i \asymp \mathrm{enc}(S_i)$ for all $1 \le i \le n$.

Here, because of the choice of the source and the target language, an encoding preserves distributability if for each sequence of distributable source term steps their simulations are pairwise distributable. In both languages two alternative steps of a term are in *conflict* with each other if—for CSP—they reduce the same action-prefix or—for CCS—they either reduce the same input using two outputs or they reduce the same output using two [replicated] inputs. Two alternative steps that are not in conflict are *distributable*.

3 Translating the CSP Synchronisation Mechanism

CSP and CCS—or the π-calculus—differ fundamentally in their communication and synchronisation mechanisms. In CSP there is only a single kind of action $c \to \cdot$, where c is a name. Synchronisation is implemented by the parallel operator $\cdot\|_A\cdot$ that in CSP is augmented with a set of names A containing the names that need to be synchronised at this point. By nesting parallel operators arbitrarily many actions on the same name can be synchronised. In CCS there are two different kinds of actions: inputs $\underline{c}$ and

outputs $\overline{c}$. Again synchronisation is implemented by the parallel operator, but in CCS only a single input and a single matching output can ever be synchronised within one step.

To encode the CSP communication and synchronisation mechanisms in CCS with name passing we make use of a technique already used in [17, 19] to translate between different variants of the π-calculus. CSP actions are translated into action announcements augmented with a lock indicating, whether the respective action was already used in the simulation of a step. The other operators of CSP are then translated into handlers for these announcements and locks. The translation of sum combines several actions under the same lock and thus ensures that only one term of the sum can ever be used. The translation of the parallel operator combines announcements of actions that need to be synchronised into a single announcement under a fresh lock, whose value is determined by the combination of the respective underlying locks at its left and right side. Announcements of actions that do not need to be synchronised are simply forwarded. A second layer—containing either a centralised or a decentralised coordinator—then triggers and coordinates the simulation of source term steps.

Action announcements are of the form $\overline{\mathsf{a}}\langle c, \mathsf{r}, \mathsf{l}, \mathsf{r}'\rangle$: c is the translation of the source term action. r is used to trigger the computation of the Boolean value of l. The lock l evaluates to $\top$ as long as the respective translated action was not successfully used in the simulation of a step. r' is used to guard the encoded continuation of the respective source term action. In the case of a successful simulation attempt involving this announcement, an output $\overline{\mathsf{r}'}\langle\top\rangle$ allows to unguard the encoded source term continuation and ensures that all following evaluations of l return $\bot$. The message $\overline{\mathsf{r}'}\langle\bot\rangle$ indicates an aborted simulation attempt and allows to restore l for later simulation attempts. Once a lock becomes $\bot$, all request for its computation return $\bot$.

Abbreviations. We introduce some abbreviations to simplify the presentation of the encodings. We use

$$[x \in A]\, P \triangleq \prod_{a \in A} [x = a]\, P$$

to test, whether an action belongs to the set of synchronised actions in the encoding of the parallel operator. As already done in [14, 15] we use Boolean-valued locks to ensure that every translation of an action is only used once to simulate a step. *Boolean locks* are channels on which only the Boolean values $\top$ (true) or $\bot$ (false) are transmitted. An unguarded output over a Boolean lock with value $\top$ represents a positive instantiation of the respective lock, whereas an unguarded output sending $\bot$ represents a negative instantiation. At the receiving end of such a channel, the Boolean value can be used to make a binary decision, which is done here within an IF-*construct*. This construct and according instantiations of locks are implemented as in [14, 15] using restriction and the order of transmitted values.

$$\overline{\mathsf{l}}\langle\top\rangle \triangleq \underline{\mathsf{l}}(t, f).\overline{t} \qquad \overline{\mathsf{l}}\langle\bot\rangle \triangleq \underline{\mathsf{l}}(t, f).\overline{f}$$

$$\underline{\mathsf{l}}(b).\text{IF } b \text{ THEN } P \text{ ELSE } Q \triangleq (\nu t, f)\left(\overline{\mathsf{l}}\langle t, f\rangle \mid \underline{t}.P \mid \underline{f}.Q\right)$$

We observe that the Boolean values $\top$ and $\bot$ are realised by a pair of links without parameters. Both cases of the IF-construct operate as guard for its subterms P and Q. The renaming policy φ reserves the names t and f to implement the Boolean values $\top$ and $\bot$.

The Algorithm. The encoding functions introduce some fresh names, that are reserved for special purposes. In Table 1 we list the reserved names $\mathscr{R}$ and provide a hint on their purpose. Moreover we reserve the names $\{ x_i \mid i \in \mathbb{N} \}$ and assume an injective mapping $\varphi' : \mathscr{X} \to \{ x_i \mid i \in \mathbb{N} \}$ that maps process variables of CSP to distinct names. The renaming policy φ for our encodings is then a function that reserves the names in $\mathscr{R} \cup \{ x_i \mid i \in \mathbb{N} \}$ and translates every source term name into three target term names. More precisely, choose $\varphi : \mathscr{N} \to \mathscr{N}^3$ such that:

1. No name is mapped onto a reserved name, i.e., $\varphi(n) \cap (\mathscr{R} \cup \{ x_i \mid i \in \mathbb{N} \}) = \emptyset$ for all $n \in \mathscr{N}$.
2. No two different names are mapped to overlapping sets of names, i.e., $\varphi(n) \cap \varphi(m) = \emptyset$ for all $n, m \in \mathscr{N}$ with $n \neq m$.

reserved names	purpose
a, a'	announce the ability to perform an action
c, c_{L}, c_{R}, z	(translated) source term channel, channel from the left/right of a parallel operator
l, l_{L}, l_{R}	lock, lock from the left/right of a parallel operator
l'	re-instantiate a positive sum lock
r, r_{L}, r_{R}	request the computation of the value of a lock
r', r'_i, r'_{L}, r'_{R}	simulate a source term step and unguard the corresponding continuations
n	order left announcements for the same channel that need to be synchronised
s, s'	distribute right announcements that need to be synchronised
b	Boolean value ($\bot$ or $\top$)
τ	fresh name used to announce τ-steps that result from concealment
once	used by the centralised encoding to avoid overlapping simulation attempts
m	fresh names used to encode internal choice
d	fresh names used to encode divergence
t, f	used to encode Boolean values

Table 1: Reserved Names.

We naturally extend the renaming policy to sets of names, i.e., $\varphi(X) \triangleq \{\, \varphi(x) \mid x \in X \,\}$ if $X \subseteq \mathcal{N}$. Let $((x_1,\ldots,x_n)).i \triangleq x_i$ denote the projection of a n-tuple to its ith element, if $1 \leq i \leq n$. Moreover $(X).i \triangleq \{\, (x).i \mid x \in X \,\}$ for a set X of n-tuples and $1 \leq i \leq n$.

The inner part of our two encodings is presented in Figure 1. The most complex case is the translation of the parallel operator $[\![P \|_A Q]\!]$ that is based on the following four steps:

Step 1: Action announcements for channels $c \notin A$

In the case of actions on channels $c \notin A$—that do not need to be synchronised here—the encoding of the parallel operator acts like a forwarder and transfers action announcements of both its subtrees further up in the parallel tree. Two different restrictions of the channel for action announcements a from the left side $[\![P]\!]$ and the right side $[\![Q]\!]$, allow to trace action announcements back to their origin as it is necessary in the following case. In the present case we use a' to bridge the action announcement over the restrictions on a.

Step 2: Action announcements for channels $c \in A$

Actions $c \in A$ need to be synchronised, i.e., can be performed only if both sides of the parallel operator cooperate on this action. Simulating this kind of synchronisation is the main purpose of the encoding of the parallel operator. The renaming policy φ translates each source term name into three target term names. The first target term name is used as reference to the original source term name and transferred in announcements. The other two names are used to simulate the synchronisation of the parallel operator in CSP. Announcements from the left are translated to outputs on the respective second name and announcements from the right to the respective third name. Restriction ensures that these outputs can only be computed by the current parallel operator encoding. The translations of the announcements into different outputs for different source term names allows us to treat announcements of different names concurrently using the term $\mathsf{Synch}(c)$, where c is a source term name.

Step 3: The term $\mathsf{Synch}(c)$

In $\mathsf{Synch}(c)$ all announcements for the same source term name c from the left are ordered in order to combine each left and each right announcement on the same name. Several such announcements

$$
\begin{aligned}
[\![P \|_A Q]\!] &\triangleq (\nu \mathsf{a}', (\varphi(A)).2, (\varphi(A)).3)\ \Big(\\
&\quad (\nu \mathsf{a})\Big([\![P]\!] \mid *\underline{\mathsf{a}}(c,\tilde{x}).\Big([c \in (\varphi(A)).1]\,\overline{(\varphi(c)).2}\langle\tilde{x}\rangle \mid [c \notin (\varphi(A)).1]\,\overline{\mathsf{a}'}\langle c,\tilde{x}\rangle\Big)\Big) \\
&\quad (\nu \mathsf{a})\Big([\![Q]\!] \mid *\underline{\mathsf{a}}(c,\tilde{x}).\Big([c \in (\varphi(A)).1]\,\overline{(\varphi(c)).3}\langle\tilde{x}\rangle \mid [c \notin (\varphi(A)).1]\,\overline{\mathsf{a}'}\langle c,\tilde{x}\rangle\Big)\Big) \\
&\quad \mid \textstyle\prod_{c\in A}\mathsf{Synch}(c) \mid *\underline{\mathsf{a}'}(\tilde{x}).\overline{\mathsf{a}}\langle\tilde{x}\rangle\Big) \\
\mathsf{Synch}(c) &\triangleq (\nu \mathsf{n})\Big(\overline{\mathsf{n}}\langle(\varphi(c)).3\rangle \\
&\quad \mid *\underline{\mathsf{n}}(\mathsf{s})\Big(\underline{(\varphi(c)).2}\big(\mathsf{r}_\mathrm{L},\mathsf{l}_\mathrm{L},\mathsf{r}'_\mathrm{L}\big).\Big((\nu \mathsf{s}')\ \Big(\\
&\qquad *\underline{\mathsf{s}}\big(\mathsf{r}_\mathrm{R},\mathsf{l}_\mathrm{R},\mathsf{r}'_\mathrm{R}\big).\big((\nu \mathsf{r},\mathsf{l},\mathsf{r}')\big(\overline{\mathsf{a}}\big\langle(\varphi(c)).1,\mathsf{r},\mathsf{l},\mathsf{r}'\big\rangle \mid \mathsf{Sim}\big) \mid \overline{\mathsf{s}'}\big\langle\mathsf{r}_\mathrm{R},\mathsf{l}_\mathrm{R},\mathsf{r}'_\mathrm{R}\big\rangle\big) \\
&\qquad \mid (\nu \mathsf{s})\big(\overline{\mathsf{n}}\langle\mathsf{s}\rangle \mid *\underline{\mathsf{s}'}(\tilde{x}).\overline{\mathsf{s}}\langle\tilde{x}\rangle\big)\ \Big)\Big)\Big)\Big) \\
\mathsf{Sim} &\triangleq (\nu \mathsf{l}')\ \Big(\overline{\mathsf{l}'} \mid *\underline{\mathsf{l}'}.\Big(\underline{\mathsf{r}}.\Big(\overline{\mathsf{r}_\mathrm{L}} \mid \underline{\mathsf{l}_\mathrm{L}}(b).\Big(\textsc{if}\ b\ \textsc{then}\ \Big(\overline{\mathsf{r}_\mathrm{R}} \mid \underline{\mathsf{l}_\mathrm{R}}(b).\Big(\textsc{if}\ b \\
&\qquad \textsc{then}\ \big(\overline{\mathsf{l}}\langle\top\rangle \mid \underline{\mathsf{r}'}(b).\big(\overline{\mathsf{r}'_\mathrm{L}}\langle b\rangle \mid \overline{\mathsf{r}'_\mathrm{R}}\langle b\rangle \mid \textsc{if}\ b\ \textsc{then}\ *\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle\bot\rangle\ \textsc{else}\ \overline{\mathsf{l}'}\big)\big) \\
&\qquad \textsc{else}\ \big(\overline{\mathsf{l}}\langle\bot\rangle \mid \overline{\mathsf{r}'_\mathrm{L}}\langle\bot\rangle\big) \mid *\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle\bot\rangle\Big)\Big) \\
&\quad \textsc{else}\ \big(\overline{\mathsf{l}}\langle\bot\rangle \mid *\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle\bot\rangle\big)\Big)\Big)\Big)\Big) \\
[\![\textstyle\sum_{i\in\mathscr{I}} c_i \to P_i]\!] &\triangleq (\nu \mathsf{r},\mathsf{l},\mathsf{r}'_1,\ldots,\mathsf{r}'_n)\ \Big(\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle\top\rangle \\
&\quad \mid \textstyle\prod_{i\in\mathscr{I}}\ \big(\overline{\mathsf{a}}\langle(c_i).1,\mathsf{r},\mathsf{l},\mathsf{r}'_i\rangle \mid *\underline{\mathsf{r}'_i}(b).\textsc{if}\ b\ \textsc{then}\ \big([\![P_i]\!] \mid *\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle\bot\rangle\big)\ \textsc{else}\ \underline{\mathsf{r}}.\overline{\mathsf{l}}\langle\top\rangle\big)\Big) \\
[\![(P)/z]\!] &\triangleq (\nu \mathsf{a}')\big((\nu \mathsf{a},z)\big([\![P]\!] \mid *\underline{\mathsf{a}}(c,\tilde{x}).\big([c = z]\,\overline{\mathsf{a}'}\langle\tau,\tilde{x}\rangle \mid [c \neq z]\,\overline{\mathsf{a}'}\langle c,\tilde{x}\rangle\big)\big) \mid *\underline{\mathsf{a}'}(\tilde{x}).\overline{\mathsf{a}}\langle\tilde{x}\rangle\big) \\
[\![f(P)]\!] &\triangleq (\nu \mathsf{a}')\ \big((\nu \mathsf{a},z)\ \big([\![P]\!] \mid *\underline{\mathsf{a}}(c,\tilde{x})\ \big(\textstyle\prod_{z/x\in f}[c = (\varphi(x)).1]\,\overline{\mathsf{a}'}\langle(\varphi(z)).1,\tilde{x}\rangle \\
&\quad \mid [c \notin \mathrm{dom}(f)]\,\overline{\mathsf{a}'}\langle c,\tilde{x}\rangle\big)\big) \mid *\underline{\mathsf{a}'}(\tilde{x}).\overline{\mathsf{a}}\langle\tilde{x}\rangle\big) \\
[\![\mathrm{DIV}]\!] &\triangleq (\nu \mathsf{d})\big(\overline{\mathsf{d}} \mid *\underline{\mathsf{d}}.\overline{\mathsf{d}}\big) \\
[\![\mu X \cdot P]\!] &\triangleq (\nu \varphi'(X))\ \Big(\overline{\varphi'(X)} \mid *\underline{\varphi'(X)}.[\![P]\!]\Big) \\
[\![X]\!] &\triangleq \overline{\varphi'(X)} \\
[\![P \sqcap Q]\!] &\triangleq (\nu \mathsf{m})(\underline{\mathsf{m}}.[\![P]\!] \mid \underline{\mathsf{m}}.[\![Q]\!] \mid \overline{\mathsf{m}}) \\
[\![\mathrm{STOP}]\!] &\triangleq \mathbf{0} \\
[\![\checkmark]\!] &\triangleq \checkmark
\end{aligned}
$$

where $\notin (\varphi(A)).1$ is short for $\in (\mathrm{fn}(P) \cup \mathrm{fn}(Q)) \setminus (\varphi(A)).1$, $\notin \mathrm{dom}(f)$ is short for $\in \mathrm{fn}(P) \setminus \mathrm{dom}(f)$, and $\neq z$ is short for $\in \mathrm{fn}(P) \setminus \{ z \}$.

Figure 1: An encoding from CSP into CCS with value passing (inner part).

may result from underlying parallel operators, sums with similar summands, and junk left over from already simulated source term steps. For each left announcement a fresh instance of s is generated and restricted. The names s and s' are used to transfer right announcements to the respective next left announcement, where s' is used to bridge over the restriction on s. This way each right announcement will eventually be transferred to each left announcement on the same name. Note that this kind of forwarding is not done concurrently but in the source language a term $P\|_A Q$ also cannot perform two steps on the same name $c \in A$ concurrently. After combining a left and a right announcement on the same source term name a fresh set of auxiliary variables $\mathsf{r}, \mathsf{l}, \mathsf{r}'$ is generated and a corresponding announcement is transmitted. The term Sim reacts to requests regarding this announcement and is used to simulate a step on the synchronised action.

Step 4: The term Sim

If a request reaches Sim it starts questioning the left and the right side. First the left side is requested to compute the current value of the lock of the action. Only if $\top$ is returned, the right side is requested to compute its lock as well. This avoids deadlocks that would result from blindly requesting the computation of locks in the decentralised encoding. If the locks of both sides are still valid the fresh lock l returns $\top$ else $\bot$ is returned. For each case Sim ensures that subsequently requests will obtain an answer by looping with $\overline{\mathsf{l}'}$ or returning $\bot$ to all requests, respectively. The messages $\overline{\mathsf{r}'_{\mathrm{L}}}\langle\bot\rangle$ and $\overline{\mathsf{r}'_{\mathrm{R}}}\langle\bot\rangle$ cause the respective underlying subterms on the left and the right side to do the same, whereas $\overline{\mathsf{r}'_{\mathrm{L}}}\langle\top\rangle$ and $\overline{\mathsf{r}'_{\mathrm{R}}}\langle\top\rangle$ cause the unguarding of encoded continuations as result of a successful simulation of a source term synchronisation step.

Basic Properties and Translated Observables. The protocol introduced by the encoding function in Figure 1 (and its outer parts introduced later) simulates a single source term step by a sequence of target term steps. Most of these steps are merely pre- and post-processing steps, i.e., they do not participate in decisions regarding the simulation of conflicting source term steps but only prepare and complete simulations. Accordingly we distinguish between *auxiliary steps*—that are pre- and post-processing steps—and *simulation steps*—that mark a point of no return by deciding which source term step is simulated. Note that the points of no return and thus the definition of auxiliary and simulation steps is different in the two variants of our encoding.

Auxiliary steps do not influence the choice of source terms steps that are simulated. Moreover they operate on restricted channels, i.e., are unobservable. Accordingly they do not change the state of the target term modulo the considered reference relations $\dot{\approx}$ and $\dot{\approx}_{\mathrm{cs}}$. We introduce some auxiliary lemmata to support this claim.

The encoding $\llbracket\cdot\rrbracket$ translates source term barbs c into free announcements with $(\varphi(c)).1$ as first value and a lock l as third value that computes to $\top$. The two coordinators, i.e., outer encodings, we introduce later, restrict the free a-channel of $\llbracket\cdot\rrbracket$.

Definition 6 (Translated Barbs)**.** Let $T \in \mathscr{P}_{\mathrm{T}}$ such that $\exists S.\ \llbracket S \rrbracket \Longrightarrow_{\mathrm{T}} T$, $\exists S.\ [\![S]\!] \Longrightarrow_{\mathrm{T}} T$, or $\exists S.\ (\!|S|\!) \Longrightarrow_{\mathrm{T}} T$. T has a translated barb c, denoted by $T \downarrow_{\llbracket c \rrbracket}$, if

- there is an unguarded output $\overline{\mathsf{a}}\langle(\varphi(c)).1, \mathsf{r}, \mathsf{l}, \mathsf{r}'\rangle$—on a free channel a in the case of $\llbracket\cdot\rrbracket$ or the outermost variant of a in the case of the later introduced encodings $[\![\cdot]\!]$ and $(\!|\cdot|\!)$—in T or
- such an announcement was consumed to unguard an IF-construct testing l and this construct is still not resolved in T

such that all locks that are necessary to instantiate l are positively instantiated.

Analysing the encoding function in Figure 1 we observe that an encoded source term has a translated barb iff the corresponding source term has the corresponding source term barb.

$$[\![P]\!] \triangleq (\nu \mathsf{a}, \mathsf{once})\big([\!|P|\!] \mid \overline{\mathsf{once}} \mid *\underline{\mathsf{once}}.\underline{\mathsf{a}}(c, \mathsf{r}, \mathsf{l}, \mathsf{r}').(\overline{\mathsf{r}} \mid \underline{\mathsf{l}}(b).(\overline{\mathsf{once}} \mid \text{IF } b \text{ THEN } \overline{\mathsf{r}'}\langle\top\rangle))\big)$$

Figure 2: A **centralised** encoding from CSP into CCS with value passing.

Observation 7. For all $S \in \mathscr{P}_{\mathrm{S}}$, it holds $S{\downarrow_c}$ iff $[\!|S|\!]{\downarrow_{[\![c]\!]}}$.

All instances of success in the translation result from success in the source. More precisely the only way to obtain $\checkmark$ in the translation is by $[\!|\checkmark|\!] \triangleq \checkmark$.

Observation 8. For all $S \in \mathscr{P}_{\mathrm{S}}$, it holds $S{\downarrow_\checkmark}$ iff $[\!|S|\!]{\downarrow_\checkmark}$.

The encoding propagates announcements through the translated parallel structure. In the translation of parallel operators it combines all left and right announcements w.r.t. to the same channel name, if this channel needs to be synchronised. Therefore we copy announcements. We use locks carrying a Boolean value to indicate whether an announcement was already used to simulate a source term step. These locks carry $\top$ in the beginning and are swapped to $\bot$ as soon as the announcement was used. In each state there is at most one positive instantiation of each lock and as soon as a lock is instantiated negatively it never becomes positive again.

Lemma 9. *Let $T \in \mathscr{P}_{\mathrm{T}}$ such that $\exists S.\ [\!|S|\!] \Longrightarrow_{\mathrm{T}} T$. Then for each variant l of the names $\mathsf{l}, \mathsf{l}_{\mathrm{L}}, \mathsf{l}_{\mathrm{R}}$*

1. *there is at most one positive instantiation of l in T,*
2. *if there is a positive instantiation of l in T then there is no other instantiation of l in T,*
3. *if there is a negative instantiation of l in T then no derivative of T contains a positive instantiation of l.*

4 The Centralised Encoding

Figure 1 describes how to translate CSP actions into announcements augmented with locks and how the other operators are translated to either forward or combine these announcements and locks. With that $[\!|\cdot|\!]$ provides the basic machinery of our encoding from CSP into CCS with name passing and matching. However it does not allow to simulate any source term step. Therefore we need a second (outer) layer that triggers and coordinates the simulation of source term steps. We consider two ways to implement this coordinator: a centralised and a decentralised coordinator. The centralised coordinator is depicted in Figure 2.

The channel once is used to ensure that simulation attempts of different source term steps cannot overlap each other. For each simulation attempt exactly one announcement is consumed. The coordinator then triggers the computation of the respective lock that was transmitted in the announcement. This request for the computation of the lock is propagated along the parallel structure induced by the translations of parallel operators until—in the leafs—encodings of sums are reached. There the request for the computation yields the transmission of the current value of the respective lock. While being transmitted back to the top of the tree, different locks that refer to synchronisation in the source terms are combined. If the computation of the lock results with $\top$ at the top of the tree, the respective source term step is simulated. Else the encoding aborts the simulation attempt and restores the consumed informations about the values of the respective locks. In both cases a new instance of $\overline{\mathsf{once}}$ allows to start the next simulation attempt. Accordingly only some post-processing steps can overlap with a new simulation attempt.

As we prove below, the points of no return in the centralised encoding can result from the consumption of action announcements by the outer encoding in Figure 2 if the corresponding lock computes to

$\top$. Moreover the encoding of internal choice and divergence introduces simulation steps, namely all steps on variants of the channels m, d, and $\varphi'(X)$. All remaining steps of the centralised encoding are auxiliary.

Definition 10 (Auxiliary and Simulation Steps)**.** A step $T \longmapsto_{\mathrm{T}} T'$ such that $\exists S \in \mathscr{P}_{\mathrm{S}}.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T$ is called a *simulation step*, denoted by $T \overset{\mapsto}{\longmapsto} T'$, if $T \longmapsto T'$ is a step on the outermost channel a and the computation of the value of the received lock l will return $\top$ or it is a step on a variant of m, d, or $\varphi'(X)$.

Else the step $T \longmapsto_{\mathrm{T}} T'$ is called an *auxiliary step*, denoted by $T \overset{\cdot}{\longmapsto} T'$.

Let $\overset{\cdot}{\Longmapsto}$ denote the reflexive and transitive closure of $\overset{\cdot}{\longmapsto}$ and let $\overset{\mapsto}{\Longmapsto} \triangleq \overset{\cdot}{\Longmapsto}\overset{\mapsto}{\longmapsto}\overset{\cdot}{\Longmapsto}$. Auxiliary steps do not change the state modulo $\dot{\approx}$.

Lemma 11. *$T \overset{\cdot}{\longmapsto} T'$ implies $T \dot{\approx} T'$ for all target terms T, T'.*

By distinguishing auxiliary and simulation steps, we can prove a condition stronger than operational correspondence, namely that each source term step is simulated by exactly one simulation step.

Lemma 12. *For all S, S', it holds $S \longmapsto_{\mathrm{S}} S'$ iff $\exists T.\ [\![S]\!] \overset{\mapsto}{\Longmapsto} T \wedge [\![S']\!] \dot{\approx} T$.*

This direct correspondence between source term steps and the points of no return of their translation allows us to prove a variant of operational correspondence that is significantly stricter than the variant proposed in [6].

Definition 13 (Operational Correspondence)**.**

An encoding $\mathrm{enc}(\cdot) : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is *operationally corresponding* w.r.t. $\dot{\approx} \subseteq \mathscr{P}_{\mathrm{T}}^2$ if it is:

Complete: $\forall S, S'.\ S \Longmapsto_{\mathrm{S}} S'$ implies $\exists T.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T \wedge [\![S']\!] \dot{\approx} T$

Sound: $\forall S, T.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T$ implies $\exists S'.\ S \Longmapsto_{\mathrm{S}} S' \wedge [\![S']\!] \dot{\approx} T$

The 'if'-part of Lemma 12 implies operational completeness w.r.t. $\dot{\approx}$ and the 'only-if'-part contains the main argument for operational soundness w.r.t. $\dot{\approx}$. Hence $[\![\cdot]\!]$ is operationally corresponding w.r.t. to $\dot{\approx}$.

Theorem 1. *The encoding $[\![\cdot]\!]$ is operationally corresponding w.r.t. to $\dot{\approx}$.*

To obtain divergence reflection we show that there is no infinite sequence of only auxiliary steps. Then divergence reflection follows from the combination of this fact and Lemma 12.

Theorem 2. *The encoding $[\![\cdot]\!]$ reflects divergence.*

The encoding function ensures that $[\![S]\!]$ has an unguarded occurrence of ✓ iff S has such an unguarded occurrence. Operational correspondence ensures that S and $[\![S]\!]$ also answer the question for the reachability of ✓ in the same way.

Theorem 3. *The encoding $[\![\cdot]\!]$ is success sensitive.*

In a similar way we can prove that a source term reaches a barb iff its translation reaches the respective translated barb.

Theorem 4. *For all S, c, it holds $S \Downarrow_c$ iff $[\![S]\!] \Downarrow_{[\![c]\!]}$.*

As proved in [18], Theorem 1, the fact that $\dot{\approx}$ is success sensitive and respects (translated) barbs, Theorem 3, and Theorem 4 imply that for all S it holds S and $[\![S]\!]$ are (success sensitive, (translated) barb respecting, weak, reduction) bisimilar, i.e., $S \dot{\approx} [\![S]\!]$. Bisimilarity is a strong relation between source terms and their translation. On the other hand, because of efficiency, distributability preserving encodings are more interesting. Because of once the encoding $[\![\cdot]\!]$ obviously does not preserve distributability. As discussed in [16] bisimulation often forbids distributed encodings. Instead they propose coupled simulation as a relation that still provides a strong connection between source terms and their translations but is more flexible. Following the approach in [16] we consider a decentralised coordinator next.

$$(\!|P|\!) \triangleq (\nu \mathsf{a})\big([\!|P|\!] \mid *\underline{\mathsf{a}}(c,\mathsf{r},\mathsf{l},\mathsf{r}').\big(\overline{\mathsf{r}} \mid \underline{\mathsf{l}}(b).\text{IF } b \text{ THEN } \overline{\mathsf{r}'}\langle\top\rangle\big)\big)$$

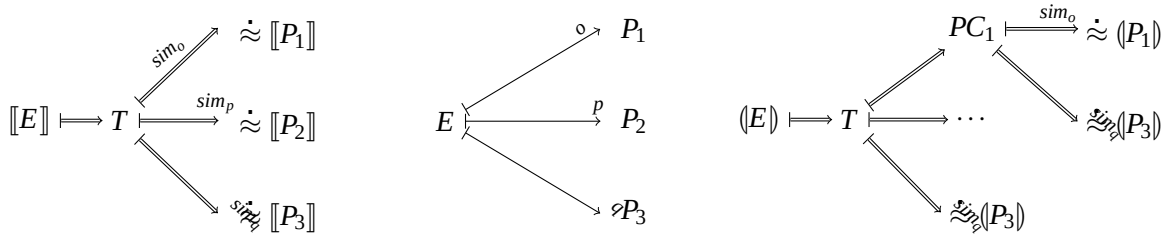
Figure 3: A **decentralised** encoding from CSP into CCS with value passing.

5 The Decentralised Encoding

Figure 3 presents a decentralised variant of the coordinator in Figure 2. The only difference between the centralised and the decentralised version of the coordinator is that the latter can request to check different locks concurrently. Technically $[\![\cdot]\!]$ and $(\!|\cdot|\!)$ differ only by the use of once. As a consequence the steps of different simulation attempts can overlap and even (pre-processing) steps of simulations of conflicting source term steps can interleave to a certain degree. Because of this effect, $(\!|\cdot|\!)$ does not satisfy the version of operational correspondence used above for $[\![\cdot]\!]$, but $(\!|\cdot|\!)$ satisfies weak operational correspondence that was proposed in [6] as part of a set of quality criteria.

Since several announcements can be processed concurrently by the decentralised coordinator, here all consumptions of announcements are auxiliary steps. Instead the consumption of positive instantiations of locks can mark a point of no return. In contrast to $[\![\cdot]\!]$ not every point of no return in $(\!|\cdot|\!)$ unambiguously marks a simulation of a single source term step, because in contrast to $[\![\cdot]\!]$ the encoding $(\!|\cdot|\!)$ introduces *partial commitments* [17, 19].

Consider the example $E = (o \to P_1 \,\square\, p \to P_2) \,||_{\{o,p\}}\, (o \to P_3 \,\square\, p \to P_4 \,\square\, q \to P_5)$.



In the example, two sides of a parallel operator have to synchronise on either action p, or action o, or action q happens without synchronisation. In the centralised encoding $[\![E]\!]$ the use of once ensures that different simulation attempts cannot overlap. Thus, only after finishing the simulation of a source term step, the simulation of another source term step can be invoked. As a consequence each state reachable from encoded source terms can unambiguously be mapped to a single state of the source term. This allows us to use a stronger version of operational correspondence and, thus, to prove that source terms and their translations are bisimilar. The corresponding 1-to-1 correspondence between source terms and their translations is visualised by the first two graphs above, where $T \,\dot{\approx}\, [\![E]\!]$.

The decentralised encoding $(\!|E|\!)$ introduces partial commitments. Assume the translation of a source term that offers several alternative ways to be reduced. Then some encodings—as our decentralised one—do not always decide on which of the source term steps should be simulated next. More precisely a partial commitment refers to a state reachable from the translation of a source term in that already some possible simulations of source term steps are ruled out, but there is still more than a single possibility left.

In the decentralised encoding announcements can be processed concurrently and parts of different simulation attempts can interleave. The only blocking part of the decentralised encoding are conflicting attempts to consume the same positive instantiation of a lock. In the presented example above there are two locks; one for each side of the parallel operator. The simulations of the step on o and p need both of

these locks, whereas to simulate the step on q only a positive instantiation of the right lock needs to be consumed. By consuming the positive instantiation of the left lock in an attempt to simulate the step on o, the simulation of the step on p is ruled out, but the simulation of the step on q is still possible. Since either the simulation of the step on o or the simulation of the step on q succeeds, the simulation of the step on p is not only blocked but ruled out. But the consumption of the instantiation of the left lock does not unambiguously decide between the remaining two simulations. The intermediate state that results from consuming the instantiation of the left lock and represents a partial commitment is visualised in the right graph above by the state PC_1.

Partial commitments forbid a 1-to-1 mapping between the states of a source term and its translations by a bisimulation. But, as shown in [16], partial commitments do not forbid to relate source terms and their translations by coupled similarity.

Whether the consumption of a positive instantiation of a lock is an auxiliary step—does not change the state of the term modulo $\dot{\approx}$—, is a partial commitment, or unambiguously marks a simulation of a single source term step depends on the surrounding term, i.e., cannot be determined without the context. For simplicity we consider all steps that reduce a positive instantiation of a lock as simulation steps. Also steps on variants of the channels m, d, and $\varphi'(X)$ are simulation steps, because they unambiguously mark a simulation of a single source term step. All remaining steps of the decentralised encoding are auxiliary.

Definition 14 (Auxiliary and Simulation Steps)**.** A step $T \longmapsto_{\mathrm{T}} T'$ such that $\exists S \in \mathscr{P}_{\mathrm{S}}.\ (\!|S|\!) \Longmapsto_{\mathrm{T}} T$ is called a *simulation step*, denoted by $T \overset{\mapsto}{\longmapsto} T'$, if $T \longmapsto T'$ reduces a positive instantiation of a lock or is a step on a variant of m, d, or $\varphi'(X)$.

Else the step $T \longmapsto_{\mathrm{T}} T'$ is called an *auxiliary step*, denoted by $T \overset{\cdot}{\longmapsto} T'$.

Again let $\overset{\cdot}{\Longmapsto}$ denote the reflexive and transitive closure of $\overset{\cdot}{\longmapsto}$ and let $\overset{\mapsto}{\Longmapsto} \triangleq \overset{\cdot}{\Longmapsto}\overset{\mapsto}{\longmapsto}\overset{\cdot}{\Longmapsto}$. Since auxiliary steps do not introduce partial commitments, they do not change the state modulo $\dot{\approx}$. The proof of this lemma is very similar to the centralised case.

Lemma 15. *$T \overset{\cdot}{\longmapsto} T'$ implies $T \dot{\approx} T'$ for all target terms T, T'.*

In contrast to the centralised encoding, the simulation of a source term step in the decentralised encoding can require more than a single simulation step and a single simulation step not unambiguously refers to the simulation of a particular source term step. The partial commitments described above forbid operational correspondence, but the weaker variant proposed in [6] is satisfied. We call this variant weak operational correspondence.

Definition 16 (Weak Operational Correspondence)**.**
An encoding $\mathrm{enc}(\cdot) : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is *weakly operationally corresponding* w.r.t. $\dot{\approx}_{\mathrm{cs}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ if it is:

Complete: $\forall S, S'.\ S \Longmapsto_{\mathrm{S}} S'$ implies $\exists T.\ (\!|S|\!) \Longmapsto_{\mathrm{T}} T \wedge (\!|S'|\!) \dot{\approx}_{\mathrm{cs}} T$

Weakly Sound: $\forall S, T.\ (\!|S|\!) \Longmapsto_{\mathrm{T}} T$ implies $\exists S', T'.\ S \Longmapsto_{\mathrm{S}} S' \wedge T \Longmapsto_{\mathrm{T}} T' \wedge (\!|S'|\!) \dot{\approx}_{\mathrm{cs}} T'$

The only difference to operational correspondence is the weaker variant of soundness that allows for T to be an intermediate state that does not need to be related to a source term directly. Instead there has to be a way from T to some T' such that T' is related to a source term.

Theorem 5. *The encoding $(\!|\cdot|\!)$ is weakly operational corresponding w.r.t. to $\dot{\approx}$.*

As in the encoding $[\![\cdot]\!]$, there is no infinite sequence of only auxiliary steps in $(\!|S|\!)$. Moreover each simulation of a source term requires only finitely many simulation steps (to consume the respective positive instantiations of locks). Thus $(\!|\cdot|\!)$ reflects divergence.

Theorem 6. *The encoding $(\!|\cdot|\!)$ reflects divergence.*

The encoding function ensures that $(\!|S|\!)$ has an unguarded occurrence of $\checkmark$ iff S has such an unguarded occurrence. Operational correspondence again ensures that S and $(\!|S|\!)$ also answer the question for the reachability of $\checkmark$ in the same way.

Theorem 7. *The encoding $(\!|\cdot|\!)$ is success sensitive.*

Similarly, a source term reaches a barb iff its translation reaches the respective translated barb.

Theorem 8. *For all S, c, it holds $S \Downarrow_c$ iff $(\!|S|\!) \Downarrow_{[\![c]\!]}$.*

Weak operational correspondence does not suffice to establish a bisimulation between source terms and their translations. But, as proved in [18], Theorem 5, the fact that $\dot{\approx}$ is success sensitive and respects (translated) observables, Theorem 7, and Theorem 8 imply that $\forall S.\ S$ and $[\![S]\!]$ are (success sensitive, (translated) barbs respecting, weak, reduction) coupled similar, i.e., $S \dot{\approx}_{\mathrm{cs}} (\!|S|\!)$.

It remains to show, that $(\!|\cdot|\!)$ indeed preserves distributability. Therefore we prove that all blocking parts of the encoding $(\!|\cdot|\!)$ refer to simulations of conflicting source term steps.

Theorem 9. *The encoding $(\!|\cdot|\!)$ preserves distributability.*

6 Conclusions

We introduced two encodings from CSP into asynchronous CCS with name passing and matching. As in [16] we had to encode the multiway synchronisation mechanism of CSP into binary communications and, similarly to [16], we did so first using a centralised controller that was then modified into a decentralised controller. By doing so we were able to transfer the observations of [16] to the present case:

1. The centralised solution allows to prove a stronger connection between source terms and their translations, namely by bisimilarity. Our decentralised solution does not relate source terms and their translations that strongly and we doubt that any decentralised solution can do so.
2. Nonetheless, decentralised solutions are possible as presented by the second encoding and they still relate source terms and their translations in an interesting way, namely by coupled similarity.

Thus as in [16] we observed a trade-off between *centralised* but *bisimilar* solutions on the one-hand side and *decentralised* but only *coupled similar* solutions on the other side.

More technically we showed here instead a trade-off between centralised but *operationally corresponding* solutions on the one-hand side and *weakly operationally corresponding* but decentralised solutions on the other side. The mutual connection between operational correspondence and bisimilarity as well as between weak operational correspondence and coupled similarity is proved in [18].

Both encodings make strict use of the renaming policy and translate into closed terms. Hence the criterion *name invariance* is trivially satisfied in both cases. Moreover we showed that both encodings are *success-sensitive*, *reflect divergence*, and even *respect barbs* w.r.t. to the standard source term (CSP) barbs and a notion of translated barbs on the target. The centralised encoding $[\![\cdot]\!]$ additionally satisfies a variant of *operational correspondence* that is stricter than the variant proposed in [6]. The decentralised encoding $(\!|\cdot|\!)$ satisfies *weak operational correspondence* as proposed in [6] and *distributability preservation* as proposed in [20]. Thus both encodings satisfy all of the criteria proposed in [6] except for compositionality. However in both cases the inner part is obviously compositional and the outer part only adds a fixed context.

References

[1] J. C. M. Baeten (2005): *A Brief History of Process Algebra*. *Theor. Comput. Sci.* 335(2-3), pp. 131–146, doi:10.1016/j.tcs.2004.07.036.

[2] E. Brinksma (1985): *A tutorial on LOTOS*. In: *Proc. of PSTV*, pp. 171–194.

[3] S. D. Brookes (1983): *On the Relationship of CCS and CSP*. In: *Proc. of ICALP, LNCS* 154, pp. 83–96, doi:10.1007/BFb0036899.

[4] H. Evrard & F. Lang (2015): *Automatic Distributed Code Generation from Formal Models of Asynchronous Concurrent Processes*. In: *Proc. of PDP*, IEEE, pp. 459–466, doi:10.1109/PDP.2015.96.

[5] R. van Glabbeek (2012): *Musings on Encodings and Expressiveness*. In: *Proc. of EXPRESS/SOS, EPTCS* 89, pp. 81–98, doi:10.4204/EPTCS.89.7.

[6] D. Gorla (2010): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. *Information and Computation* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[7] M. Hatzel, C. Wagner, K. Peters & U. Nestmann (2015): *Encoding CSP into CCS (Extended Version)*. Technical Report. Available at `http://arxiv.org/abs/1508.01127`.

[8] C. A. R. Hoare (1978): *Communicating Sequential Processes*. *Communications of the ACM* 21(8), pp. 666–677, doi:10.1145/359576.359585.

[9] C. A. R. Hoare (2006): *Why ever CSP? Electronic Notes in Theoretical Computer Science* 162(0), pp. 209–215, doi:10.1016/j.entcs.2006.01.031.

[10] I. Lanese & U. Montanari (2006): *Hoare vs Milner: Comparing Synchronizations in a Graphical Framework With Mobility*. *Electronic Notes in Theoretical Computer Science* 154(2), pp. 55 – 72, doi:10.1016/j.entcs.2005.03.032.

[11] R. Milner (1980): *A calculus of communicating systems*. Springer, doi:10.1007/3-540-10235-3.

[12] R. Milner (1986): *Process Constructors and Interpretations (Invited Paper)*. In: *IFIP Congress*, pp. 507–514.

[13] R. Milner & D. Sangiorgi (1992): *Barbed Bisimulation*. In: *Proc. of ICALP, LNCS* 623, pp. 685–695, doi:10.1007/3-540-55719-9_114.

[14] U. Nestmann (1996): *On Determinacy and Nondeterminacy in Concurrent Programming*. Ph.D. thesis, Universität Erlangen-Nürnberg.

[15] U. Nestmann & B. C. Pierce (2000): *Decoding Choice Encodings*. *Information and Computation* 163(1), pp. 1–59, doi:10.1006/inco.2000.2868.

[16] J. Parrow & P. Sjödin (1992): *Multiway synchronization verified with coupled simulation*. In: *Proc. of CONCUR, LNCS* 630, Springer, pp. 518–533, doi:10.1007/bfb0084813.

[17] K. Peters (2012): *Translational Expressiveness*. Ph.D. thesis, TU Berlin.

[18] K. Peters & R. van Glabbeek (2015): *Analysing and Comparing Encodability Criteria*. In: *Proc. of EXPRESS/SOS*, EPTCS 190, pp. 46–60, doi:10.4204/EPTCS.190.4.

[19] K. Peters & U. Nestmann (2012): *Is it a "Good" Encoding of Mixed Choice?* In: *Proc. of FoSSaCS, LNCS* 7213, pp. 210–224, doi:10.1007/978-3-642-28729-9_14.

[20] K. Peters, U. Nestmann & U. Goltz (2013): *On Distributability in Process Calculi*. In: *Proc. of ESOP, LNCS* 7792, Springer, pp. 310–329, doi:10.1007/978-3-642-37036-6_18.

[21] P. Sjödin (1991): *From LOTOS Specifications to Distributed Implementations*. Ph.D. thesis, Department of Computer Science, Uppsala University. Available as Report DoCS 91/31.

Encoding the Factorisation Calculus
(Representing the Intensional in the Extensional)

Reuben N. S. Rowe
Department of Computer Science
University College London
r.rowe@ucl.ac.uk

Jay and Given-Wilson have recently introduced the Factorisation (or SF-) calculus as a minimal fundamental model of *intensional* computation. It is a combinatory calculus containing a special combinator, F, which is able to examine the internal structure of its first argument. The calculus is significant in that as well as being combinatorially complete it also exhibits the property of structural completeness, i.e. it is able to represent any function on terms definable using pattern matching on arbitrary normal forms. In particular, it admits a term that can decide the structural equality of any two arbitrary normal forms.

Since SF-calculus is combinatorially complete, it is clearly at least as powerful as the more familiar and paradigmatic Turing-powerful computational models of λ-calculus and Combinatory Logic. Its relationship to these models in the converse direction is less obvious, however. Jay and Given-Wilson have suggested that SF-calculus is strictly more powerful than the aforementioned models, but a detailed study of the connections between these models is yet to be undertaken.

This paper begins to bridge that gap by presenting a faithful encoding of the Factorisation Calculus into the λ-calculus preserving both reduction and strong normalisation. The existence of such an encoding is a new result. It also suggests that there is, in some sense, an equivalence between the former model and the latter. We discuss to what extent our result constitutes an equivalence by considering it in the context of some previously defined frameworks for comparing computational power and expressiveness.

1 Introduction

Mathematical models of computation are useful in studying the formal properties of programming practice. Indeed, the field of computing today arose partly out of the study of such abstract models: namely Turing Machines [33], the λ-calculus [8], and Combinatory Logic [10], which are consequently considered to be archetypal computational models. It is standard practice to qualify the abilities, or expressiveness, of a formal model of computation by demonstrating that it may *simulate* (and be simulated by) the operation of other formal models. This is the very essence of the notion of *Turing-completeness*, which encapsulates the intuition that a model may carry out any operation that is 'effectively computable'. To construct such a simulation one must first give an injective mapping, showing how the terms of the source model may be represented by terms of the target. For example, this is the basis behind the process of Gödelization and the Church encoding of natural numbers [22]. Two basic properties are then required: that each atomic operational step of the source model is reflected by one or more steps of the target, and that a program of the target model *terminates* whenever the corresponding source program does. The former is a key ingredient of Landin's influential work on comparing languages [25], while the latter is used as a criterion for comparing expressiveness by, e.g., Felleisen [12]. Formal definitions of encodings incorporating these properties, referred to as "faithful", are already in use by the 90s, e.g. in [2], and are now common (see e.g. [15]).

S. Crafa and D. Gebler (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2015)
EPTCS ??, 2015, pp. 1–15, doi:10.4204/EPTCS.??.??

© Reuben N. S. Rowe
This work is licensed under the Creative Commons Attribution License.

Recently, Jay's work on formal models of generic pattern matching [17] have led, in collaboration with Given-Wilson, to the formulation of the *Factorisation Calculus*. This is a combinatory calculus comprising two combinators: the S combinator, familiar from Combinatory Logic; and a new F combinator. The purpose of the latter is to enable arbitrary (head) normal terms to be *factorised*, that is split into their constituent parts, thereby allowing the examination of the internal structure of terms. This endows Factorisation Calculus with an interesting and powerful property: that of *structure completeness*. This means that any function on terms themselves definable by pattern matching over *arbitrary* normal forms is *representable*. Thus, Factorisation Calculus can be viewed as a minimal, fundamental model characterising not only the abstract notion of *pattern matching* but also *intensional computation*.

Jay and Given-Wilson show that Factorisation Calculus is Turing-complete, demonstrating a straightforward simulation of Combinatory Logic in their calculus. Moreover, due to its structural completeness, there is a term of Factorisation Calculus which can decide the structural equality of any two arbitrary normal forms. Conversely, factorisation and structural equality of normal forms *cannot* be so represented in λ-calculus and Combinatory Logic, thus these models are *not* structure complete. This hints at some sort of disparity in the expressivity of the two models. In their original and subsequent research [18, 14, 19], Jay and Given-Wilson speculate that the added expressive power may manifest itself in a non-existence result for simulations of Factorisation Calculus in λ-calculus, but this is not pursued in detail.

We show that there *does* exist a simulation of Factorisation Calculus within λ-calculus. The existence of such a simulation has not been demonstrated before, and this is the primary contribution of our paper. The simulation is made possible by a construction due to Berarducci and Böhm, which shows how to encode a certain class of term rewriting systems within λ-calculus. We show that Factorisation Calculus can be simulated by such a term rewriting system, whence the result follows. In the classical framework, our result signifies that Factorisation Calculus is no more powerful than λ-calculus. Thus there appears to be a mismatch between our result and the structure completeness property that the standard simulation-based notion of equivalence does not account for. To begin to try and resolve this, we consider some research in the literature which refines the concept of computational equivalence and discuss how our result relates to this.

Outline The rest of this paper is organised as follows. Section 2 recalls Jay and Given-Wilson's Factorisation Calculus and its basic properties. Section 3 describes Berarducci and Böhm's construction for encoding so-called canonical rewrite systems within the λ-calculus. In Section 4 we present our technical contribution: a simulation of the Factorisation Calculus in the λ-calculus via this construction. Section 5 then discusses, in light of our results, how the relative expressiveness of Factorisation Calculus and λ-calculus may be characterised. Section 6 concludes and remarks on areas for future work.

2 Factorisation Calculus

We begin by presenting Jay and Given-Wilson's Factoriation Calculus itself, and review its principal properties. Factorisation Calculus, or more accurately SF-calculus[1], is a combinatory calculus whose terms are those of the free algebra over the two-element signature containing the combinators S and F, which each reduce upon being applied to three arguments. The former is the familiar combinator from Combinatory Logic [10] which applies its first and second arguments to duplicates of its third. The F combinator, on the other hand, introduces new capabilities in the form of *factorisation*: it is

[1] We may say that any combinatory calculus that is structure complete is *a* factorisation calculus.

able to examine the *internal* structure of its first argument and process its second and third in different ways depending on whether that argument is *atomic* (i.e. itself a combinator) or *compound* (i.e. a partial application). To illustrate this, consider how the F combinator reduces in the following two instances:

$$\mathsf{F}\,\mathsf{S}\,M\,N \to M \qquad\qquad \mathsf{F}\,(\mathsf{S}\,X)\,M\,N \to N\,\mathsf{S}\,X$$

Observe that when the first argument is atomic, it eliminates its third argument and returns its second. On the other hand, when the first argument is compound it eliminates its second argument and *factorises* the first into its left- and right-hand constituent components, passing these *separately* to its third argument.

Formally, SF-calculus is defined as follows.

Definition 2.1 (SF-calculus [18, § 4])**.** *The SF-calculus is a combinatory rewrite system over terms (ranged over by uppercase roman letters M, N, etc.) given by the following grammar:*

$$M, N ::= \mathsf{S} \mid \mathsf{F} \mid MN$$

Terms of the form S, F, $\mathsf{S}M$, $\mathsf{F}M$, $\mathsf{S}MN$, *or* $\mathsf{F}MN$ *(i.e. partially applied combinators) are called* factorable forms*. Reduction of terms is the smallest contextually closed binary relation $\to_{SF}$ on terms (with the reflexive transitive closure denoted by $\to^*_{SF}$) satisfying:*

$$\begin{aligned}
\mathsf{S}MNX &\to_{SF} MX\,(NX) && \\
\mathsf{F}OMN &\to_{SF} M && \text{if } O \text{ is } \mathsf{S} \text{ or } \mathsf{F} \\
\mathsf{F}\,(PQ)\,MN &\to_{SF} NPQ && \text{if } PQ \text{ is a factorable form}
\end{aligned}$$

Reduction of SF-calculus is confluent, and the K combinator of Combinatory Logic can be represented in SF-calculus by $\mathsf{F}\,\mathsf{F}$ (also, indeed, by $\mathsf{F}\,\mathsf{S}$). Thus, there is a trivial encoding of Combinatory Logic in SF-calculus which preserves reduction and strong normalisation [14].

The behaviour of the F combinator gives SF-calculus an *intensional* quality: one may define higher order functions in SF-calculus which discriminate between functions whose *implementations* are different even when those functions are *extensionally* equal (i.e. produce identical outputs for identical inputs). For example, for any normal form X, the term $I_X \equiv \mathsf{S}\,(\mathsf{F}\,\mathsf{F})\,X$ implements the identity function (i.e. $I_X\,M \to^*_{\mathsf{SF}} M$ for all M) and thus all such terms are extensionally equal. However, an SF-term T can be constructed which distinguishes them (by behaving as $T\,I_X \to^*_{\mathsf{SF}} X$). Moreover, one can construct an SF-term that can decide the equality of any two *arbitrary* normal forms.

The intensional behaviour of SF-calculus is formally characterised by a property called *structure completeness*, which captures the notion that every *symbolic computation* (i.e. Turing-computable symbolic function) on normal forms is represented by some term.

Definition 2.2 (Structure Completeness [18, § 7-8])**.** *Let $\mathscr{C}$ be a confluent combinatory calculus whose terms include variables, with reduction relation $\to^*_{\mathscr{C}}$. Define* patterns *to be the* linear *normal forms (i.e. containing no more than one occurrence of each variable), and* matchable forms *to be partially applied combinators.*

1. *A* match $\{U/P\}$ *of a pattern P against a term U may be defined to* succeed *with a substitution of terms for variables, or* fail *as follows (where* Id *denotes the identity function and $\uplus$ the disjoint union of substitutions with match failure as an absorbing element):*

$$\begin{aligned}
\{U/x\} = [U/x] \qquad \{A/A\} = \mathsf{Id} \qquad & \text{(if } A \text{ atomic)} \\
\{UV/PQ\} = \{U/P\} \uplus \{V/Q\} \qquad & \text{(if } UV \text{ a compound)} \\
\{U/P\} = \mathsf{fail} \qquad & \text{(otherwise, if } U \text{ matchable)}
\end{aligned}$$

2. *A* case *is an equation of the form $P = M$ where P is a pattern and M an arbitrary term, which defines a* symbolic function $\mathscr{G}$ *on terms by $\mathscr{G}(U) = \sigma(M)$ if $\{U/P\}$ succeeds with substitution σ, and $\mathscr{G}(U) = U$ if it fails.*
3. *A confluent combinatory calculus is* structure complete *if for every pattern P and term M, there is some term G such that $GU \to^*_{\mathscr{C}} \mathscr{G}(U)$ for every term U on which $\mathscr{G}$ is defined; i.e. the symbolic function defined by every case is represented by some term.*

Structure completeness subsumes combinatorial completeness since $\lambda x.M$ is given by the case $x = M$.

Theorem 2.3 ([18, Cor. 8.4])**.** *SF-calculus is structure complete.*

The F combinator itself represents a symbolic computation $\mathscr{F}$, namely that of factorisation:

$$\begin{aligned} \mathscr{F}(A, M, N) &= M && \text{if } A \text{ is atomic} \\ \mathscr{F}(PQ, M, N) &= NPQ && \text{if } PQ \text{ is compound} \end{aligned}$$

A significant (and arguably remarkable) fact is that $\mathscr{F}$ cannot be represented in Combinatory Logic (for definitions of atomic and compound appropriate thereto).

Theorem 2.4 ([18, Thm. 3.2])**.** *Factorisation of SK-combinators is a symbolic computation that is not representable in Combinatory Logic.*

The equality predicate on normal forms also has no representation in Combinatory Logic. Thus, there exist (symbolic) functions, which are clearly 'computable' from an empirical point of view, that are not (directly) representable in Combinatory Logic (there is also a similar result for λ-calculus [4]). This result clearly points towards some form of added expressivity possessed by SF-calculus over the archetypal computational models. It is to this issue that we will return in Section 5.

3 Strongly Normalising Solutions of Equational Systems in λ-calculus

We now reiterate the interpretation result of Berarducci and Böhm [5], upon which our technical contribution rests. Essentially, this result says that systems of equations for a particular class of term algebras can be given solutions in the λ-calculus such that the representation of each atomic term of the algebra is *strongly normalising* thus having a *normal form*. Moreover, when the set of equations is interpreted as a rewrite system the encoding of terms preserves reduction and strong normalisation.

We assume the usual definitions of the λ-calculus without further explanation (readers may refer to [4] for details), with Λ denoting the set of lambda terms, $\to^*_\beta$ denoting the (multi-step) β-reduction relation, and $=_\beta$ denoting β-equality (i.e. the equivalence relation on lambda terms induced by β-reduction). Furthermore, we also assume the familiar algebraic notion of the set $\mathrm{Ter}(\Sigma)$ of (Σ-)*terms* over the signature (set of *function symbols*, each with an associated arity) Σ. We can then also consider the set $\Lambda(\Sigma)$ of *extended* lambda terms (i.e. lambda terms which may contain Σ-terms); notice that both $\mathrm{Ter}(\Sigma)$ and Λ are (strict) subsets of $\Lambda(\Sigma)$.

Definition 3.1 (Canonical Systems of Equations)**.** *Fix a signature Σ and let $\mathscr{E}$ be a set of equations between terms $t \in \mathrm{Ter}(\Sigma)$. We say that $\mathscr{E}$ is* canonical *if Σ can be partitioned into two disjoint subsets Σ_0 and Σ_1 (i.e. $\Sigma = \Sigma_0 \cup \Sigma_1$) such that: each equation in $\mathscr{E}$ is of the form $f(c(x_1,\ldots,x_m),y_1,\ldots,y_n) = t$ with $c \in \Sigma_0$ and $f \in \Sigma_1$ and where the variables $x_1,\ldots,x_m,y_1,\ldots,y_n$ are all distinct and form a superset of the variables in the term t; and for each distinct pair $(c,f) \in \Sigma_0 \times \Sigma_1$ there is at most one equation in $\mathscr{E}$ of this form. We say that $\mathscr{E}$ is* complete *if for each distinct pair $(c,f) \in \Sigma_0 \times \Sigma_1$ there is exactly one such equation in $\mathscr{E}$.*

Canonical systems of equations, then, partition the signature into a set Σ_0 of (algebraic datatype) *constructors*, and Σ_1 of *programs* defined by pattern matching over the constructors on the first argument. Notice that any incomplete canonical system of equations can trivially be made complete by adding equations for the missing cases which simply project one of the function's arguments[2].

As an example of a canonical system of equations, we may observe that the usual recursive definition of addition over the datatype of (Peano) natural numbers is such a system:

$$\mathsf{add}\,(\mathsf{zero},x) = x \qquad \mathsf{add}\,(\mathsf{succ}\,(x),y) = \mathsf{succ}\,(\mathsf{add}\,(x,y))$$

We have a signature containing one function symbol add, one nullary constructor zero, and one unary constructor succ; moreover in this simple case, the equation system is already complete. In fact, every partial recursive function (on natural numbers) can be defined by a canonical system of equations [5, 6].

Given an equational system $\mathscr{E}$ over a signature Σ, we can also take it to define a term rewriting system on $\mathrm{Ter}(\Sigma)$ by reading each equation as a *rewrite* rule, i.e. $f_i(c_j(x_1,\ldots,x_m),y_1,\ldots,y_n) \to t$. We will write $\to_{\mathscr{E}}$ for the (one-step) reduction relation of the rewrite system defined by $\mathscr{E}$ in this way (i.e. the smallest binary relation on terms satisfying the rewrite rules and closed under substitution and contexts), and $\to^*_{\mathscr{E}}$ for its reflexive, transitive closure (i.e. multi-step reduction). Ultimately, the aim is to interpret equational systems (and their associated rewrite systems) within λ-calculus.

Definition 3.2 (Interpretations)**.** *A* representation *of the signature Σ is a function $\phi : \Sigma \to \Lambda$ from the function symbols of Σ to (closed) lambda terms, and induces a map $(\cdot)^\phi : \Lambda(\Sigma) \to \Lambda$ in the obvious way, namely by $x^\phi = x$, $(\lambda x.M)^\phi = \lambda x.M^\phi$, $(MN)^\phi = M^\phi N^\phi$, and for $f \in \Sigma$, $f\,(t_1,\ldots,t_n)^\phi = \phi(f)\,{t_1}^\phi \ldots {t_n}^\phi$. We say that a representation ϕ* satisfies *(or* solves*) $\mathscr{E}$ if for each equation $t_1 = t_2$ (and corresponding rewrite rule $t_1 \to t_2$) in $\mathscr{E}$ we have ${t_1}^\phi =_\beta {t_2}^\phi$ (and correspondingly also ${t_1}^\phi \to^*_\beta {t_2}^\phi$). When a representation ϕ satisfies $\mathscr{E}$, we say that ϕ is an* interpretation *(or a* solution*) of $\mathscr{E}$ within λ-calculus.*

The following construction gives a special kind of representation for canonical systems of equations.

Definition 3.3 (Canonical Representations)**.** *Let $\mathscr{E}$ be a canonical system of equations that partitions the signature Σ into constructors $\Sigma_0 = \{c_1,\ldots,c_r\}$ and programs $\Sigma_1 = \{f_1,\ldots,f_k\}$. Without loss of generality we may assume that $\mathscr{E}$ is complete, and so for each $1 \le i \le k$ and $1 \le j \le r$ let $b_{(i,j)}$ denote the term t such that $f_i(c_j(x_1,\ldots,x_m),y_1,\ldots,y_n) = t \in \mathscr{E}$.*

We will make use of the following notational abbreviations:

- *Let $\langle t_1,\ldots,t_n\rangle$ denote the Church n-tuple, i.e. $\lambda x.xt_1\ldots t_n$.*
- *Let Π^n_k (where $1 \le k \le n$) be the n-ary k^{th} projection function, i.e. $\lambda x_1\ldots x_n.x_k$.*
- *For $k \ge i > 1$, let $t_i,\ldots,t_k,\ldots,t_{i-1}$ denote the cyclic permutation of $t_1,t_2,\ldots,t_k$ beginning with t_i; (in an abuse of notation we may also take $t_i,\ldots,t_k,\ldots,t_{i-1} = t_1,t_2,\ldots,t_k$ when $i = 1$).*

We now define two disjoint representations ϑ and ζ for constructors and programs respectively.

(Representation of Constructors) *For each $1 \le i \le r$, we define the representation of the constructor c_i as follows (where n is the arity of c_i):*

$$\vartheta(c_i) = \lambda x_1\ldots x_n\,f.\,f\Pi^r_i x_1\ldots x_n f$$

(Representation of Programs) *We choose k distinct fresh variables $v_1,\ldots,v_k$ not occurring in $\mathscr{E}$ and fix a 'pre-representation', ψ, of Σ_1 defined by $\psi(f_i) = \langle v_i,\ldots,v_k,\ldots,v_{i-1}\rangle$ for each $1 \le i \le k$.*

[2] Alternatively, one might want to introduce a new nullary constructor (denoting an 'error' value) and add equations for the missing cases that simply return this value.

Using this representation, and the representation of constructors defined above, we then define $k \times r$ lambda terms $t_{(i,j)}$ ($1 \leq i \leq k$, $1 \leq j \leq r$), using the equations in $\mathscr{E}$ as follows:

$$t_{(i,j)} = \lambda x_1 \ldots x_m v_i \ldots v_k \ldots v_{i-1} y_1 \ldots y_n . (b_{(i,j)}{}^{\psi})^{\vartheta}$$

where $f_i(c_j(x_1,\ldots,x_m),y_1,\ldots,y_n) = b_{(i,j)} \in \mathscr{E}$ is the equation defining the behaviour of f_i when given a datum constructed using c_j as its first argument. We now define k terms, each one a Church r-tuple collating the bodies of all the cases for one of the programs in Σ_1, as follows:

$$t_i = \langle t_{(i,1)}, \ldots, t_{(i,r)} \rangle$$

(where $1 \leq i \leq k$). Each program is then represented by a Church k-tuple containing the collated representations of each program defintion, beginning with its own. That is, ζ is defined by:

$$\zeta(f_i) = \langle t_i, \ldots, t_k, \ldots, t_{i-1} \rangle \quad (1 \leq i \leq k)$$

The representation $\phi = \vartheta \cup \zeta$ is called a canonical *representation of Σ with respect to $\mathscr{E}$.*

To gain some insight into the construction defined above, one can observe that it is related to an encoding of data attributed to Scott[3] (and thus commonly referred to in the literature as the Scott encoding), which has subsequently been developed by others (e.g. [30, 27, 16, 31]). In the more familiar 'standard' encoding of functions, a fixed-point combinator is used to solve any recursion in the definition. This has the effect of making recursion *explicit*, and thus the representations of recursive functions have infinite expansions consisting of a 'list' of distinct instances of the function body, one for each recursive call that may be made. Applying the function to a datum then corresponds to a *fold* of the datum over this list, which discards the remaining infinity of recursive calls once the base case is reached. Therefore, as described by Böhm et al. [6, §3], in this scheme functions are 'diverging objects which, when applied to data, may "incidentally" converge'. In encodings of the Scott variety, the recursive nature of functions is kept *implicit* and, while still triggered by application to a datum, only reproduced 'on demand'. Hence we obtain finite objects which now 'may "incidentally" diverge' when applied to data[4].

To explicate the particular encoding specified by Definition 3.3, we point out that the representation of a constructor is a (lambda) function that takes in the appropriate number of arguments (the *sub-data* of the datum that is subsequently constructed) and then waits to be given a function, which will be the program to be executed. Now, looking at how the constructor representation uses this function argument, we see that programs should expect to be given a projection function, followed by a number of sub-data, and then they are also passed *a copy of themselves*. It is this final element which is the key to Scott-type encodings, and allows recursion to be kept implicit. Looking now at the representation of programs we see that they are Church *k*-tuples containing an element for each program defined by $\mathscr{E}$ (each of which is a Church *r*-tuple, where each element is a representation of one of the cases of that program's definition). Thus the representation of each program contains the definition of *every* program defined by $\mathscr{E}$; in particular it will contain the definition of each program which it may itself invoke. To illustrate in more detail how the encoding works, we can consider the general reduction sequence of a term representing the application of some program prog_i to some arguments, the first of which is a datum constructed as $c_j(d_1,\ldots,d_m)$:

$$(\mathsf{prog}_i\,(c_j\,(d_1,\ldots,d_m))\,\mathsf{arg}_1 \ldots \mathsf{arg}_m)^{\phi} = \langle t_i,\ldots,t_{i-1}\rangle\,(c_j\,(d_1,\ldots,d_m))^{\phi}\,\mathsf{arg}_1^{\phi} \ldots \mathsf{arg}_m^{\phi}$$

[3] The citation can be found in Curry, Hindley and Seldin [11, p. 504].

[4] This reversed form of the slogan is also due to Böhm et al., and illustrates the dual nature of the Scott and Church encodings.

$$\rightarrow^*_\beta \quad (\lambda x.x t_i \ldots t_{i-1})\,(\lambda f.f\Pi^r_j d_1 \ldots d_m f)\,\mathsf{arg}^\phi_1 \ldots \mathsf{arg}^\phi_m \tag{1}$$

$$\rightarrow_\beta \quad (\lambda f.f\Pi^r_j d_1 \ldots d_m f)\,t_i \ldots t_{i-1}\,\mathsf{arg}^\phi_1 \ldots \mathsf{arg}^\phi_m \tag{2}$$

$$\rightarrow_\beta \quad t_i\,\Pi^r_j d_1 \ldots d_m\,t_i\,t_{i+1} \ldots t_{i-1}\,\mathsf{arg}^\phi_1 \ldots \mathsf{arg}^\phi_m \tag{3}$$

$$= \quad \langle t_{i,1}, \ldots, t_{i,r}\rangle\,\Pi^r_j d_1 \ldots d_m\,t_i\,t_{i+1} \ldots t_{i-1}\,\mathsf{arg}^\phi_1 \ldots \mathsf{arg}^\phi_m \tag{4}$$

$$= \quad (\lambda x.x t_{i,1} \ldots t_{i,r})\,\Pi^r_j d_1 \ldots d_m\,t_i\,t_{i+1} \ldots t_{i-1}\,\mathsf{arg}^\phi_1 \ldots \mathsf{arg}^\phi_m \tag{5}$$

$$\rightarrow_\beta \quad \Pi^r_j\,t_{i,1} \ldots t_{i,r}\,d_1 \ldots d_m\,t_i\,t_{i+1} \ldots t_{i-1}\,\mathsf{arg}^\phi_1 \ldots \mathsf{arg}^\phi_m \tag{6}$$

$$\rightarrow_\beta \quad t_{i,j}\,d_1 \ldots d_m\,t_i\,t_{i+1} \ldots t_{i-1}\,\mathsf{arg}^\phi_1 \ldots \mathsf{arg}^\phi_m \tag{7}$$

$$= \quad (\lambda x_1 \ldots x_m v_i \ldots v_k \ldots v_{i-1} y_1 \ldots y_n.({b_{(i,j)}}^\psi)^\vartheta)\,d_1 \ldots d_m\,t_i\,t_{i+1} \ldots t_{i-1}\,\mathsf{arg}^\phi_1 \ldots \mathsf{arg}^\phi_m$$

$$\rightarrow^*_\beta \quad (b_{(i,j)}\,[d_1/x_1, \ldots, d_m/x_m, \mathsf{arg}_1/y_1, \ldots, \mathsf{arg}_n/y_n])^\phi \tag{8}$$

When the program is applied to a datum (Eq. (1)), its representation arranges to apply the datum first to the representations of each program beginning with its own, and then to the remainder of the arguments (Eq. (2)). Then, the particular structure of the datum will reduce the expression to pick out the appropriate case of the program definition to be executed, and apply it to the sub-data and the representations of each program, having duplicated the program being executed (Eqs. (3) to (7)). This then reduces to the representation of the appropriate substitution instance of the function body (Eq. (8)).

The result of Berarducci and Böhm says that a canonical representation gives an interpretation that also preserves strong normalisation.

Theorem 3.4 (Interpretation Theorem [5, Thm 3.4])**.** *Let Σ be a signature and $\mathscr{E}$ a canonical set of equations for Σ; then any canonical representation ϕ for Σ with respect to $\mathscr{E}$ is an interpretation of $\mathscr{E}$ within λ-calculus. In addition $(\cdot)^\phi$ preserves strong normalisation of closed terms.*

4 Encoding the Factorisation Calculus

In this section, we present our novel technical contribution: an encoding of SF-calculus in λ-calculus. We believe that this is the first such encoding presented in the literature. Our encoding is a faithful simulation; it preserves both the reduction behaviour of terms (thus also β-equality) and their termination behaviour (i.e. strong normalisation). In this section we shall make use of standard notation and results for term rewriting systems, details of which may be found in [24].

The key step to the encoding is to define a rewrite system behaviourally equivalent to SF-calculus that is also canonical, in the sense of Definition 3.1. It is then simply a matter of applying the construction of Berarducci and Böhm to obtain the encoding. Thus it is our translation of SF-calculus into this intermediate rewrite system that is the primary novelty of our contribution.

We are aiming to derive a set of rewrite rules that is *canonical* and so we must translate the schematic definition of Jay and Given-Wilson, as presented in Section 2, into one consisting of algebraic rewrite rules. There are two salient features of Definition 3.1 that we must take into account: that the rewrite rules must make a distinction between *programs* and *constructors*; and that the left-hand side of each rewrite rule must contain exactly one program symbol and one constructor. To obtain rewrite rules of the required form, we recast SF-calculus as a *curryfied, applicative* term rewriting system. That is, we first introduce an explicit program symbol app to denote application and use the symbols S and F solely as *constructors*. Secondly we stratify the combinators $\mathsf{C} \in \{\mathsf{S}, \mathsf{F}\}$ into sets $\{\mathsf{C}_0, \mathsf{C}_1, \mathsf{C}_2\}$ of constructors, each of which represent successive *partial* applications of their underlying combinator C. Although this

'currying' process is well-known from the world of functional programming, the reader may refer to [20, 3] for a formal definition of this process in the context of general term rewriting.

We may take the rewrite rules for the S combinator directly from the standard curryfied applicative formulation of Combinatory Logic (see e.g. [3]):

$$\mathsf{app}(\mathsf{S}_0, x) \to \mathsf{S}_1(x) \quad \mathsf{app}(\mathsf{S}_1(x), y) \to \mathsf{S}_2(x, y) \quad \mathsf{app}(\mathsf{S}_2(x, y), z) \to \mathsf{app}(\mathsf{app}(x, z), \mathsf{app}(y, z))$$

The rules for producing the partial applications of the F combinator are similarly straightforward:

$$\mathsf{app}(\mathsf{F}_0, x) \to \mathsf{F}_1(x) \qquad \mathsf{app}(\mathsf{F}_1(x), y) \to \mathsf{F}_2(x, y)$$

The rewrite rule for the full application of the F combinator is more tricky because we must find a way of implementing its two possible reductions. As in the original formulation of SF-calculus, since the choice of which reduction to make is determined by the structure of the first argument we should like to be able to use the pattern-matching capabilities inherent in the term rewriting discipline, e.g. by giving rewrite rules such as:

$$\mathsf{app}(\mathsf{F}_2(\mathsf{S}_0, y), z) \to y \qquad \mathsf{app}(\mathsf{F}_2(\mathsf{F}_1(x), y), z) \to \mathsf{app}(\mathsf{app}(z, \mathsf{F}_0), x)$$

However these rules are *not* canonical since they contain two occurrences of a constructor: they are pattern-matching 'too deeply'. We can circumvent this by introducing an auxiliary *program* symbol $\mathsf{f\text{-}reduce}$ and then having the rewrite rule for the F_2 case of app delegate to this new program:

$$\mathsf{app}(\mathsf{F}_2(x, y), z) \to \mathsf{f\text{-}reduce}(x, y, z)$$

Since $\mathsf{f\text{-}reduce}$ is an independent program symbol, and only needs to pattern match on its first argument to determine which result to compute, we may give canonical rewrite rules for it, such as the following:

$$\mathsf{f\text{-}reduce}(\mathsf{S}_0, y, z) \to y \qquad \mathsf{f\text{-}reduce}(\mathsf{F}_1(x), y, z) \to \mathsf{app}(\mathsf{app}(z, \mathsf{F}_0), x)$$

We now have all the components to be able to present a canonical rewrite system that faithfully implements SF-calculus.

Definition 4.1 (Currified Applicative SF-Calculus)**.** *Let $\Sigma_{SF} = \Sigma_0 \cup \Sigma_1$ be the signature comprising the set $\Sigma_0 = \{\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{F}_0, \mathsf{F}_1, \mathsf{F}_2\}$ of constructors and the set $\Sigma_1 = \{\mathsf{app}, \mathsf{f\text{-}reduce}\}$ of programs.* Curryfied Applicative SF-calculus *is the term rewriting system $\mathsf{SF}^{\mathscr{C}}_{@}$ defined by the rewrite rules given in Figure 1 over the signature Σ_{SF}. We denote its one-step and many-step reduction relations by $\to_{\mathsf{SF}^{\mathscr{C}}_{@}}$ and $\to^*_{\mathsf{SF}^{\mathscr{C}}_{@}}$.*

Notice that the rewrite rules of $\mathsf{SF}^{\mathscr{C}}_{@}$ are canonical, in the sense of Definition 3.1, and that they are also *complete*. We also remark that $\mathsf{SF}^{\mathscr{C}}_{@}$ is an *orthogonal* term rewriting system [24, Def. 2.1.1].

It is interesting to observe that our implementation of SF-calculus as a canonical applicative term rewriting system has involved an application of the *Visitor* design pattern[5] [13]. When it comes to reducing a complete application of the F combinator, the computation must proceed based on the particular identity of some object (i.e. the first argument), but *without having any knowledge of that identity*. The solution is to apply the visitor pattern, which involves invoking a new 'visit' operation (that we call $\mathsf{f\text{-}reduce}$) on the object, which in response executes the appropriate behaviour based on its *self*-knowledge of its own identity. We do not think it is entirely coincidental that the visitor pattern has arisen in our work: its connection with structural matching has already been noted [28], and investigating this connection further is an avenue for future research.

There is a straightforward translation from SF-calculus to $\mathsf{SF}^{\mathscr{C}}_{@}$.

[5] In fact, the encoding that we are presenting in this paper arose as a direct result of considering how the Factorisation Calculus could be implemented in (Featherweight) Java.

$$\begin{array}{ll}
\mathsf{app}(\mathsf{S}_0, x) \to \mathsf{S}_1(x) & \mathsf{app}(\mathsf{F}_0, x) \to \mathsf{F}_1(x) \\
\mathsf{app}(\mathsf{S}_1(x), y) \to \mathsf{S}_2(x, y) & \mathsf{app}(\mathsf{F}_1(x), y) \to \mathsf{F}_2(x, y) \\
\mathsf{app}(\mathsf{S}_2(x, y), z) \to \mathsf{app}(\mathsf{app}(x, z), \mathsf{app}(y, z)) & \mathsf{app}(\mathsf{F}_2(x, y), z) \to \mathsf{f\text{-}reduce}(x, y, z)
\end{array}$$

$$\begin{array}{ll}
\mathsf{f\text{-}reduce}(\mathsf{S}_0, y, z) \to y & \mathsf{f\text{-}reduce}(\mathsf{S}_1(x), y, z) \to \mathsf{app}(\mathsf{app}(z, \mathsf{S}_0), x) \\
\mathsf{f\text{-}reduce}(\mathsf{F}_0, y, z) \to y & \mathsf{f\text{-}reduce}(\mathsf{F}_1(x), y, z) \to \mathsf{app}(\mathsf{app}(z, \mathsf{F}_0), x)
\end{array}$$

$$\begin{array}{l}
\mathsf{f\text{-}reduce}(\mathsf{S}_2(p, q), y, z) \to \mathsf{app}(\mathsf{app}(z, \mathsf{app}(\mathsf{S}_0, p)), q) \\
\mathsf{f\text{-}reduce}(\mathsf{F}_2(p, q), y, z) \to \mathsf{app}(\mathsf{app}(z, \mathsf{app}(\mathsf{F}_0, p)), q)
\end{array}$$

Figure 1: A Complete Set of Canonical Rewrite Rules for Currified Applicative SF-calculus

Definition 4.2 (Translation of SF-calculus to $\mathsf{SF}^{\mathscr{C}}_{@}$)**.** *The translation $[\![\cdot]\!]_@$ from SF-terms to $SF^{\mathscr{C}}_{@}$-terms is defined by $[\![\mathsf{S}]\!]_@ = \mathsf{S}_0$, $[\![\mathsf{F}]\!]_@ = \mathsf{F}_0$, and $[\![MN]\!]_@ = \mathsf{app}([\![M]\!]_@, [\![N]\!]_@)$.*

We now show that $\mathsf{SF}^{\mathscr{C}}_{@}$ faithfully implements SF-calculus.

Lemma 4.3 ($[\![\cdot]\!]_@$ Preserves Reduction)**.** *Let M and N be SF-terms; if $M \to^*_{SF} N$ then $[\![M]\!]_@ \to^*_{SF^{\mathscr{C}}_{@}} [\![N]\!]_@$.*

Proof. It is sufficient to consider the basic reduction rules of SF-calculus. In the interests of clarity, we underline the redex that is contracted at each step. The case for S is straightforward:

$$\begin{array}{rl}
[\![\mathsf{S}MNX]\!]_@ = \mathsf{app}(\mathsf{app}(\underline{\mathsf{app}(\mathsf{S}_0, [\![M]\!]_@)}, [\![N]\!]_@), [\![X]\!]_@) & \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\underline{\mathsf{app}(\mathsf{S}_1([\![M]\!]_@), [\![N]\!]_@)}, [\![X]\!]_@) \\
& \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \underline{\mathsf{app}(\mathsf{S}_2([\![M]\!]_@, [\![N]\!]_@), [\![X]\!]_@)} \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\mathsf{app}([\![M]\!]_@, [\![X]\!]_@), \mathsf{app}([\![N]\!]_@, [\![X]\!]_@)) \\
& = [\![MX(NX)]\!]_@
\end{array}$$

The case for F with S the first argument (i.e. atomic) is as follows (the other atomic case is symmetric):

$$\begin{array}{rl}
[\![\mathsf{F}\mathsf{S}MN]\!]_@ = \mathsf{app}(\mathsf{app}(\underline{\mathsf{app}(\mathsf{F}_0, \mathsf{S}_0)}, [\![M]\!]_@), [\![N]\!]_@) & \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\underline{\mathsf{app}(\mathsf{F}_1(\mathsf{S}_0), [\![M]\!]_@)}, [\![N]\!]_@) \\
& \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \underline{\mathsf{app}(\mathsf{F}_2(\mathsf{S}_0, [\![M]\!]_@), [\![N]\!]_@)} \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \underline{\mathsf{f\text{-}reduce}(\mathsf{S}_0, [\![M]\!]_@, [\![N]\!]_@)} \to_{\mathsf{SF}^{\mathscr{C}}_{@}} [\![M]\!]_@
\end{array}$$

When the first argument to F is a factorable form, we must further consider its structure. The case for when the first argument is $\mathsf{S}X$ (for some term X) is as follows:

$$\begin{array}{l}
[\![\mathsf{F}(\mathsf{S}X)MN]\!]_@ = \mathsf{app}(\mathsf{app}(\mathsf{app}(\mathsf{F}_0, \underline{\mathsf{app}(\mathsf{S}_0, [\![X]\!]_@)}), [\![M]\!]_@), [\![N]\!]_@) \\
\quad \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\mathsf{app}(\underline{\mathsf{app}(\mathsf{F}_0, \mathsf{S}_1([\![X]\!]_@))}, [\![M]\!]_@), [\![N]\!]_@) \\
\quad \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\underline{\mathsf{app}(\mathsf{F}_1(\mathsf{S}_1([\![X]\!]_@)), [\![M]\!]_@)}, [\![N]\!]_@) \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \underline{\mathsf{app}(\mathsf{F}_2(\mathsf{S}_1([\![X]\!]_@), [\![M]\!]_@), [\![N]\!]_@)} \\
\quad \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \underline{\mathsf{f\text{-}reduce}(\mathsf{S}_1([\![X]\!]_@), [\![M]\!]_@, [\![N]\!]_@)} \to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\mathsf{app}([\![N]\!]_@, \mathsf{S}_0), [\![X]\!]_@) = [\![N\mathsf{S}X]\!]_@
\end{array}$$

Again, the case for when the first argument is $\mathsf{F}X$ (for some term X) is symmetric and can be obtained from the above sequence by replacing each occurrence of S_0 by F_0 and each occurrence of S_1 by F_1.

The case for when the first argument is $\mathsf{S}XY$ (for some terms X and Y) is as follows:

$$[\![\mathsf{F}(\mathsf{S}XY)MN]\!]_@ = \mathsf{app}(\mathsf{app}(\mathsf{app}(\mathsf{F}_0, \mathsf{app}(\underline{\mathsf{app}(\mathsf{S}_0, [\![X]\!]_@)}, [\![Y]\!]_@)), [\![M]\!]_@), [\![N]\!]_@)$$

$$\begin{aligned}
&\to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\mathsf{app}(\mathsf{app}(\mathsf{F}_0, \underline{\mathsf{app}(\mathsf{S}_1(\llbracket X\rrbracket_@), \llbracket Y\rrbracket_@)}), \llbracket M\rrbracket_@), \llbracket N\rrbracket_@)\\
&\to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\mathsf{app}(\underline{\mathsf{app}(\mathsf{F}_0, \mathsf{S}_2(\llbracket X\rrbracket_@, \llbracket Y\rrbracket_@))}, \llbracket M\rrbracket_@), \llbracket N\rrbracket_@)\\
&\to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\underline{\mathsf{app}(\mathsf{F}_1(\mathsf{S}_2(\llbracket X\rrbracket_@, \llbracket Y\rrbracket_@)), \llbracket M\rrbracket_@)}, \llbracket N\rrbracket_@)\\
&\to_{\mathsf{SF}^{\mathscr{C}}_{@}} \underline{\mathsf{app}(\mathsf{F}_2(\mathsf{S}_2(\llbracket X\rrbracket_@, \llbracket Y\rrbracket_@), \llbracket M\rrbracket_@), \llbracket N\rrbracket_@)}\\
&\to_{\mathsf{SF}^{\mathscr{C}}_{@}} \underline{\mathsf{f\text{-}reduce}(\mathsf{S}_2(\llbracket X\rrbracket_@, \llbracket Y\rrbracket_@), \llbracket M\rrbracket_@, \llbracket N\rrbracket_@)}\\
&\to_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathsf{app}(\mathsf{app}(\llbracket N\rrbracket_@, \mathsf{app}(\mathsf{S}_0, \llbracket X\rrbracket_@)), \llbracket Y\rrbracket_@) = \llbracket N\,(\mathsf{S}\,X)\,Y\rrbracket_@
\end{aligned}$$

Once more, the case for when the first argument is $\mathsf{F}\,X\,Y$ (for some terms X and Y) is symmetric and can be obtained from the above sequence by replacing each occurrence of S_0 by F_0, each occurrence of S_1 by F_1, and each occurrence of S_2 by F_2. □

To show that $\llbracket\cdot\rrbracket_@$ preserves strong normalisation, we will rely on the notion of a *perpetual reduction sequence*. We recall the relevant definitions of perpetual reductions and their properties [21]. A term t is called an ∞-*term* (also denoted $\infty(t)$) if it has an infinite reduction sequence. A reduction *step* $t \to s$ is called *perpetual* if $\infty(t)$ implies $\infty(s)$, that is it preserves divergence, and a reduction sequence $t_1 \to \ldots \to t_n$ is perpetual if every step $t_i \to t_{i+1}$ is perpetual. Clearly, a perpetual reduction sequence $t \to^* t'$ also preserves divergence. A *redex* u is called perpetual if its contraction in every context yields a perpetual reduction step. Let $u \to t$ be a substitution instance of a rewrite rule r (so u is a redex and t its r-contraction), then call the subterms of u that are those substituted for the variables in r the *arguments* of u. Such an argument is said to be *erased* if it corresponds to a variable that does *not* occur in the right-hand side of r. It is the case that for orthogonal rewrite systems every redex whose erased arguments are strongly normalising and closed (i.e. containing no variables) is perpetual [21, Cor. 5.1].

We first prove a couple of auxiliary lemmas.

Lemma 4.4. *Let N be an SF-normal form, then $\llbracket N\rrbracket_@$ is strongly normalising.*

Proof. We characterise the normal forms as terms taking one of the following forms: S, F, $\mathsf{S}\,X$, $\mathsf{F}\,X$, $\mathsf{S}\,X\,Y$ or $\mathsf{F}\,X\,Y$, in which each subterm is also a normal form. We then proceed by induction on the size of terms. The base cases, i.e. when N is either S of F are trivial since then $\llbracket N\rrbracket_@$ is itself a normal form. For the inductive cases, notice that the terms $\mathsf{S}_1(\llbracket X\rrbracket_@)$, $\mathsf{F}_1(\llbracket X\rrbracket_@)$, $\mathsf{S}_2(\llbracket X\rrbracket_@, \llbracket Y\rrbracket_@)$ and $\mathsf{F}_2(\llbracket X\rrbracket_@, \llbracket Y\rrbracket_@)$ are strongly normalising since they are head normal and by induction $\llbracket X\rrbracket_@$ and $\llbracket Y\rrbracket_@$ are strongly normalising as X and Y are by definition normal forms (smaller than N). It is then straightforward to show in each case that the (unique) reduction from $\llbracket N\rrbracket_@$ to its corresponding head normal form given above is perpetual since it does not erase any arguments, and $\mathsf{SF}^{\mathscr{C}}_{@}$ is an orthogonal rewrite system. The result then follows. □

Lemma 4.5. *Let X, M and N be SF-normal forms, and O an operator (i.e. either S or F) such that $O\,X\,M\,N \to_{SF} R$; then $\mathscr{C}[\llbracket O\,X\,M\,N\rrbracket_@] \to^*_{\mathsf{SF}^{\mathscr{C}}_{@}} \mathscr{C}[\llbracket R\rrbracket_@]$ is a perpetual reduction sequence for any $\mathsf{SF}^{\mathscr{C}}_{@}$-term context $\mathscr{C}$.*

Proof. We show that there is a reduction sequence $\llbracket O\,X\,M\,N\rrbracket_@ \to^*_{\mathsf{SF}^{\mathscr{C}}_{@}} \llbracket R\rrbracket_@$ which contracts a perpetual redex at each step, from which the result immediately follows. In fact, the reduction sequences that witness this are exactly those that are used to show preservation of reduction. In each case notice that, in the reduction sequence demonstrated in the proof of Lemma 4.3, the only erased argument (when it exists) is in the final reduction step and in each case this argument is either $\llbracket M\rrbracket_@$ or $\llbracket N\rrbracket_@$ which by Lemma 4.4 is strongly normalising since M and N are normal forms (and also closed since we do not

consider SF-terms with variables). Thus, since $\mathsf{SF}^{\mathscr{C}}_{@}$ is an orthogonal rewrite system, it follows that the redex contracted at each step is perpetual. □

We can now prove the following result.

Lemma 4.6 ($[\![\cdot]\!]_@$ Preserves Strong Normalisation)**.** *Let M be a strongly normalising SF-term; then $[\![M]\!]_@$ is strongly normalising.*

Proof. We use the same technique as used in the proof of [5, Thm. 3.4(2)], and proceed by (strong) induction on the length n of the longest reduction sequence from M to its normal form. When $n = 0$, then we have that M is a (SF-)normal form and thus $[\![M]\!]_@$ is strongly normalising w.r.t $\to^*_{\mathsf{SF}^{\mathscr{C}}_{@}}$ by Lemma 4.4. When $n > 0$ then M must contain at least one redex $OXYZ$. Consider an *innermost* redex, whose contractum is the term R. Thus $M = \mathscr{C}[OXYZ] \to_{\mathsf{SF}} \mathscr{C}[R] = N$ (for some (SF-)term context $\mathscr{C}$) and X, Y and Z are normal forms (since the redex is innermost). Now, since M is strongly normalising so too is N, and the length of its longest reduction sequence must be strictly less than n (otherwise n would not be maximum). Thus by the inductive hypothesis $[\![N]\!]_@$ is strongly normalising. Consider now the structure of $[\![M]\!]_@$: we have $[\![M]\!]_@ = \mathscr{C}'[[\![OXYZ]\!]_@]$ for some ($\mathsf{SF}^{\mathscr{C}}_{@}$-)term context $\mathscr{C}'$. Since X, Y and Z are normal forms, by Lemma 4.5 there is a perpetual reduction sequence from $[\![M]\!]_@ = \mathscr{C}'[[\![OXYZ]\!]_@]$ to $[\![N]\!]_@ = \mathscr{C}'[[\![R]\!]_@]$, i.e. one which preserves divergence. Therefore, since $[\![N]\!]_@$ is strongly normalising so too is $[\![M]\!]_@$ (if it were not, neither would $[\![N]\!]_@$ be). □

Using Berarducci and Böhm's construction, outlined in Section 3, we obtain an encoding of SF-calculus in the λ-calculus.

Definition 4.7 (Translation of SF-calculus to λ-calculus)**.** *Fix a canonical representation ϕ_{SF} of Σ_{SF} w.r.t. the rewrite rules of $\mathsf{SF}^{\mathscr{C}}_{@}$. The mapping $[\![\cdot]\!]_\lambda = (\cdot)^{\phi_{SF}} \circ [\![\cdot]\!]_@$ translates SF-calculus to λ-calculus.*

We leave it as an exercise to the reader to compute such a canonical representation ϕ_{SF}.

We now present our main result: that $[\![\cdot]\!]_\lambda$ is a faithful encoding of SF-calculus in λ-calculus.

Theorem 4.8 (Faithful Encoding of SF-calculus in λ-calculus)**.** *The translation $[\![\cdot]\!]_\lambda$ of SF-calculus into λ-calculus preserves reduction and strong normalisation.*

Proof. The result follows directly from the fact that the two translations that are composed to obtain $[\![\cdot]\!]_\lambda$, namely $[\![\cdot]\!]_@$ and $(\cdot)^{\phi_{\mathsf{SF}}}$, each satisfy both these properties. In the case of the former, we refer to Lemmas 4.3 and 4.6; for the latter to the results of Berarducci and Böhm, cf. Theorem 3.4 (note that all terms are closed, since we do not consider SF-calculus with variables). □

5 Expressiveness of Factorisation: Discussion & Related Work

We now turn out attention to the question of the expressiveness of SF-calculus relative to λ-calculus and Combinatory Logic. On one hand our results, i.e. Theorem 4.8, along with those of Jay and Given-Wilson [18], show that SF-calculus and λ-calculus simulate the same executions. On the other, SF-calculus is structure complete whereas λ-calculus is not. Is this a contradiction and, if so, how may it be resolved? Notwithstanding the long tradition of using simulations to characterise computational equivalence, it has since been realised that a refinement of this notion is necessary to draw richer, more meaningful comparisons. While a complete and in-depth analysis is not within the scope of the current paper, by discussing our results with reference to some of this work we aim to draw some concrete conclusions about the how the expressiveness of structure completeness and SF-calculus may be characterised.

The 'Standard' Notion of Equivalence. From the earliest research into computability, two aspects of abstract notions of computation were identified as relevant to the idea of expressiveness. Firstly, one wants to compare the respective set of *functions* that each model computes. For example, in recursion theory it was shown that the set of primitive recursive functions (on natural numbers) is a strict subset of the recursive functions (see e.g. [32]). At the same time, there is a requirement to compare models that operate, at a fundamental level, in diverse domains. Turing Machines, λ-calculus and Combinatory Logic are, operationally, quite different ways to compute. It was shown however that via particular (and now canonical) representations of numbers, each model can 'compute' the same set of functions on natural numbers, namely the partial recursive functions. Conversely, Gödelization allows each of the computations in these models to be *simulated* by a partial recursive function [22].

This idea of simulation extends to the operation of the models themselves: each model may simulate the operational behaviour of the others. Such simulations also appear to abstract away the problem of *representation*: often we do want to compute functions of natural numbers, however more often we desire to compute functions over different domains; it is incumbent upon us to represent elements of the desired domain of discourse as terms of the computational model. The initial characterisation of the set of 'computable' functions was over the domain of natural numbers[6], but what is the set of 'computable' functions over some other given domain? With simulations between models it seems that one may at least lay this question aside by observing that whatever can be represented in one model may then also be represented in the other, and therefore whatever functions they do compute it is the same in both cases. A stronger conclusion would be that the set of computable functions over arbitrary domains is isomorphic to the computable functions on natural numbers.

This simulation method has long since become the standard: to show two models are of equivalent computational power, demonstrate simulations of each in the other. One model of computation is only more powerful than another, then, if it is *not* possible to simulate the former in the latter. The approach has been cemented over the years, notably in Landin's now seminal work [25]. As the search space of formal computation has been explored the notion of simulation has been adapted accordingly, and there are now a number of sophisticated simulations between all sorts of models, both sequential and concurrent (e.g. [1, 29]). In this tradition, Theorem 4.8 is a result showing that λ-calculus is computationally *as powerful as* SF-calculus.

Refining the Notion of Expressiveness. It was already noted over two decades ago that despite the broad applicability and application of the simulation method, it is not actually a fine-grained enough notion to provide a complete and universal characterisation of expressiveness. Felleisen observed that since the languages we wish to compare are (usually) Turing-complete, other methods (than simulation) must be found in order to verify claims of relative (in-)expressiveness [12]. He proposed a framework based on the concept, from logic, of *eliminability* of symbols from conservative extensions [23]. One logical system $\mathscr{L}$ is a conservative extension of another system $\mathscr{L}'$ if the expressions (formulae) and theorems of the latter are subsets of those of the former. A symbol of $\mathscr{L}$ (which is not in $\mathscr{L}'$) is eliminable if there is a homomorphism (i.e. a map preserving the syntactic structure) $\varphi : \mathsf{Exp}(\mathscr{L}) \to \mathsf{Exp}(\mathscr{L}')$ from the expressions of $\mathscr{L}$ to the expressions of $\mathscr{L}'$, which acts as identity for expressions of $\mathscr{L}'$, such that an expression t is a theorem of $\mathscr{L}$ if and only if $\varphi(t)$ is a theorem of $\mathscr{L}'$. Felleisen extends this to programming languages by analogy - formulae are (syntactically valid) programs and the theorems are the terminating programs. Then we may say that language $\mathscr{L}$ is *more* expressive than language $\mathscr{L}'$ when

[6] Turing's work is different in this respect, since he deliberately embarked on a characterisation of computable functions over a different domain, namely that of strings of arbitrary symbols.

the former adds some *non-eliminable* syntactic construct, i.e. one which cannot be 'translated away'. Thus the standard simulation approach is refined by imposing an extra criterion when one language is a superset of another: can the larger one be built from the smaller one using *macros*?

To place SF-calculus in this framework we can consider SKF-calculus, i.e. the extension of SF-calculus by including an additional atomic term: the familiar K combinator. Since SKF-calculus is a proper extension of Combinatory Logic, obtained by adding the F combinator, we can apply the expressiveness test of Felleisen. That is, we ask is F eliminable? The answer to this question is *no*; indeed this is guaranteed by Theorem 2.4. In this sense, Jay and Given-Wilson's results *do* justify the claim that SF-calculus is more expressive than Combinatory Logic and λ-calculus.

More Abstract Notions of Computational Equivalence. The simulation method, and its refinement described above, are still firmly grounded in an *operational* view of computation, but recent work has sought to anchor formal comparisons of expressiveness in a more general, *abstract* basis. A notable contribution to this effort is the work of Boker and Dershowitz [7]. They abstract the notion of computational model as simply its *extension*, i.e. the set of functions over its inherent domain that it computes, and consider simulations (encodings) between them. They derive the remarkable result that combining the standard simulation approach with the natural containment of one extensionality within another leads to a paradox: some computational models can simulate models which are *strictly* more powerful in the sense that they have larger extensionalities (i.e. compute more functions). Thus, some representations *add more computational power*.

This result begs the question: do we consider simulations via such 'active' mappings to constitute an equivalence? One may suspect that the problem lies in allowing *injective* mappings between domains, and that imposing stricter conditions (e.g. bijections) would ensure 'passiveness'. This is indeed the case, but adopting such restrictions is useless for most comparisons since the passive encodings are ones which are "almost identity". We have no choice but to allow such encodings, although we do have the option of considering a hierarchy of equivalences based on the properties of the encodings used. Boker and Dershowitz define four increasingly stricter notions of (in-)equivalence based on, respectively: injective encodings (power equivalence), corresponding to the standard approach; injective encodings for which the images are computable in the simulating models (decent power equivalence); bijective encodings (bijective power equivalence); and bijections that are inverses of each other (isomorphism). Boker and Dershowitz also show that there are models (including Turing Machines and the recursive numeric functions) which cannot simulate any stronger models; they are *interpretation complete*.

We may also gain insight into the relative expressiveness of SF-calculus and λ-calculus using this framework. Our result shows that they are *power equivalent*. Theorem 2.4 shows that they cannot be bijectively power equivalent (nor, therefore, isomorphic) since that would imply that λ-calculus could distinguish arbitrary normal forms. Thus, in this sense they are not equivalent. We do not know if SF-calculus is *bijectively stronger* than λ-calculus; this would require demonstrating a bijective encoding of the latter in the former. Also, we do not know if λ-calculus is decently power equivalent to SF-calculus; we consider it at least a possibility. We would also expect that, due to its intensional capabilities, SF-calculus is interpretation complete.

Related to the work of Boker and Dershowitz, is that of Cockett and Hofstra [9], and Longley [26] which are both concerned with category-theoretic descriptions of abstract computational models. In these frameworks model equivalence is interpreted by categorical isomorphism, and so akin to the strongest notion of equivalence considered by Boker and Dershowitz.

6 Conclusions & Future Work

In this paper, we have considered the relationship of the recently introduced SF-calculus to the 'canonical' computational model of λ-calculus and, so by extension, Combinatory Logic. We have demonstrated that SF-calculus can be faithfully encoded (i.e. simulated) in the λ-calculus by defining a behaviourally equivalent applicative term rewriting system and then interpreting this system in λ-calculus using a construction of Berarducci and Böhm. This result shows that SF-calculus and λ-calculus are of equivalent computational power, according to the classical interpretation of computational equivalence. We have also considered the relationship of SF-calculus to the λ-calculus using a more nuanced interpretation of equivalence, informed by research in the literature. Moreover, we hope to have exposed both SF-calculus and Berarducci and Böhm's encoding to greater prominence. We feel that they are both subjects of great interest which deserve to be better known.

With respect to future work, there is still great scope for investigating the expressiveness of SF-calculus. There are the open questions we have highlighted regarding SF-calculus as it relates to the framework of Boker and Dershowitz. The categorical and denotational natures of SF-calculus also deserve exploration. Beyond this, the wider question of how best to qualify computational expressiveness still remains; we believe the study of SF-calculus can provide further insights on this. For example, the structural completeness property is already a new metric; Jay and Vergara have also considered a strengthening of the notion of decent power equivalence in which the simulation of each model in itself via the composition of the two encodings is computable in each model [19]. Quantitative questions regarding expressiveness also present themselves: e.g. our encoding of SF-calculus in λ-calculus leads to a large increase in the size of terms; are there lower bounds on the size of such increases which meaningfully quantify expressive power?

Acknowledgements We would like to thank Barry Jay, and also the anonymous reviewers for helpful comments in the preparation of the final version of this paper. This research was supported by EPSRC Grant EP/K040049/1.

References

[1] M. Abadi & L. Cardelli (1996): *A Theory Of Objects*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[2] Martín Abadi, Luca Cardelli & Ramesh Viswanathan (1996): *An Interpretation of Objects and Object Types*. In: *Principles of Programming Languages (POPL'96)*, pp. 396–409, doi:10.1145/237721.237809.

[3] S. van Bakel & M. Fernández (1997): *Normalization Results for Typeable Rewrite Systems*. *Information and Computation* 133(2), pp. 73–116, doi:10.1006/inco.1996.2617.

[4] H. Barendregt (1981): *The Lambda Calculus, Its Syntax and Semantics*. North-Holland, Amsterdam.

[5] Alessandro Berarducci & Corrado Böhm (1992): *A Self-Interpreter of Lambda Calculus Having a Normal Form*. In: *Computer Science Logic, 6th Workshop, CSL '92*, pp. 85–99, doi:10.1007/3-540-56992-8_7.

[6] Corrado Böhm, Adolfo Piperno & Stefano Guerrini (1994): *Lambda-Definition of Function(al)s by Normal Forms*. In: *ESOP'94, Edinburgh, U.K.*, pp. 135–149, doi:10.1007/3-540-57880-3_9.

[7] Udi Boker & Nachum Dershowitz (2009): *The Influence of Domain Interpretations on Computational Models*. *Applied Mathematics and Computation* 215(4), pp. 1323–1339, doi:10.1016/j.amc.2009.04.063.

[8] Alonzo Church (1936): *An Unsolvable Problem of Elementary Number Theory*. *American Journal of Mathematics* 58(2), pp. 345–363, doi:10.2307/2371045.

[9] J.R.B. Cockett & Pieter J.W. Hofstra (2010): *Categorical Simulations*. *Journal of Pure and Applied Algebra* 214(10), pp. 1835 – 1853, doi:10.1016/j.jpaa.2009.12.028.

[10] H. B. Curry & R. Feys (1958): *Combinatory Logic*. 1, North-Holland, Amsterdam.

[11] H. B. Curry, J. R. Hindley & J. P. Seldin (1972): *Combinatory Logic*. 2, North-Holland, Amsterdam.

[12] Matthias Felleisen (1991): *On the Expressive Power of Programming Languages*. *Sci. Comput. Program.* 17(1-3), pp. 35–75, doi:10.1016/0167-6423(91)90036-W.

[13] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides (1995): *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[14] Thomas Given-Wilson (2014): *Expressiveness via Intensionality and Concurrency*. In: *Theoretical Aspects of Computing - ICTAC 2014, Bucharest, Romania*, pp. 206–223, doi:10.1007/978-3-319-10882-7_13.

[15] Daniele Gorla (2010): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. *Inf. Comput.* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[16] J.M. Jansen, P. W. M. Koopman & R. Plasmeijer (2006): *Efficient Interpretation by Transforming Data Types and Patterns to Functions*. In: *Trends in Functional Programming, Nottingham, UK*, pp. 73–90.

[17] Barry Jay (2009): *Pattern Calculus - Computing with Functions and Structures*. Springer.

[18] Barry Jay & Thomas Given-Wilson (2011): *A Combinatory Account of Internal Structure*. *J. Symb. Log.* 76(3), pp. 807–826, doi:10.2178/jsl/1309952521.

[19] Barry Jay & Jose Vergara (2014): *Confusion in the Church-Turing Thesis*. Available at http://arxiv.org/abs/1410.7103. CoRR abs/1410.7103.

[20] Stefan Kahrs (1995): *Confluence of Curried Term-Rewriting Systems*. *J. Symb. Comput.* 19(6), pp. 601–623, doi:10.1006/jsco.1995.1035.

[21] Zurab Khasidashvili, Mizuhito Ogawa & Vincent van Oostrom (2001): *Perpetuality and Uniform Normalization in Orthogonal Rewrite Systems*. *Inf. Comput.* 164(1), pp. 118–151, doi:10.1006/inco.2000.2888.

[22] S. C. Kleene (1936): *Lambda-definability and recursiveness*. *Duke Mathematical Journal* 2, pp. 340–353, doi:10.1215/S0012-7094-36-00227-2.

[23] S.C. Kleene (1952): *Introduction to Metamathematics*. Bibliotheca Mathematica, Wolters-Noordhoff.

[24] J. W. Klop (1992): *Term Rewriting Systems*. In S. Abramsky, Dov M. Gabbay & S. E. Maibaum, editors: *Handbook of Logic in Computer Science (Vol. 2)*, OUP, Inc., New York, NY, USA, pp. 1–116.

[25] Peter J. Landin (1966): *The Next 700 Programming Languages*. *Commun. ACM* 9(3), pp. 157–166, doi:10.1145/365230.365257.

[26] John Longley (2014): *Computability Structures, Simulations and Realizability*. *Mathematical Structures in Computer Science* 24(2), doi:10.1017/S0960129513000182.

[27] Torben Æ. Mogensen (1992): *Efficient Self-Interpretation in Lambda Calculus*. *Journal of Functional Programming* 2, pp. 345–364, doi:10.1017/S0956796800000423.

[28] Jens Palsberg & C. Barry Jay (1998): *The Essence of the Visitor Pattern*. In: *COMPSAC '98, August 19-21, 1998, Vienna, Austria*, pp. 9–15, doi:10.1109/CMPSAC.1998.716629.

[29] D. Sangiorgi & D. Walker (2001): *PI-Calculus: A Theory of Mobile Processes*. CUP, New York, NY, USA.

[30] J. Steensgaard-Madsen (1989): *Typed Representation of Objects by Functions*. *ACM Transactions on Programming Languages and Systems* 11(1), pp. 67–89, doi:10.1145/59287.77345.

[31] Aaron Stump (2009): *Directly Reflective Meta-Programming*. *Higher-Order and Symbolic Computation* 22(2), pp. 115–144, doi:10.1007/s10990-007-9022-0.

[32] G.J. Tourlakis (1984): *Computability*. Reston Publishing Company.

[33] A. M. Turing (1937): *On Computable Numbers, with an Application to the Entscheidungsproblem*. *Proceedings of the London Mathematical Society* s2-42(1), pp. 230–265, doi:10.1112/plms/s2-42.1.230.