BRICS

Basic Research in Computer Science

BRICS RS-99-42 Nestmann & Pierce: Decoding Choice Encodings

Decoding Choice Encodings

Uwe Nestmann
Benjamin C. Pierce

BRICS Report Series **RS-99-42**

ISSN 0909-0878 **December 1999**

See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

http://www.brics.dk
ftp://ftp.brics.dk
This document in subdirectory RS/99/42/

Decoding Choice Encodings*

Uwe Nestmann, BRICS†, Aalborg University, Denmark
Benjamin C. Pierce, University of Pennsylvania, USA

Abstract

We study two encodings of the *asynchronous π-calculus* with *input-guarded choice* into its choice-free fragment. One encoding is divergence-free, but refines the atomic commitment of choice into gradual commitment. The other preserves atomicity, but introduces divergence. The divergent encoding is fully abstract with respect to weak bisimulation, but the more natural divergence-free encoding is not. Instead, we show that it is fully abstract with respect to *coupled simulation*, a slightly coarser—but still coinductively defined—equivalence that does not enforce bisimilarity of internal branching decisions. The correctness proofs for the two choice encodings introduce a novel proof technique exploiting the properties of explicit *decodings* from translations to source terms.

*This paper is a revised full version of a technical report that appeared first as IMMD-VII-01/96 at University of Erlangen and TR 342 at University of Cambridge. An extended summary then appeared in the Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96) LNCS 1119, pages 179–194, Springer-Verlag, 1996. The work was mainly carried out while the first author was at the University of Erlangen-Nürnberg, Germany (supported by *Deutsche Forschungsgemeinschaft*, Sonderforschungsbereich 182, project C2, and by *Deutscher Akademischer Austauschdienst* within the ARC-program), and while the second author was at the University of Cambridge, UK (supported by the *British Science and Engineering Research Council*). The first revision took place while the first author was supported by a post-doc fellowship from ERCIM (*European Research Consortium for Informatics and Mathematics*); it is available as ERCIM Research Report 10/97-R051.

†Basic Research in Computer Science, Centre of the Danish National Research Foundation.

Contents

1 Introduction

The problem of implementing the concurrent choice operator in terms of lower-level constructs is interesting from a number of points of view. Theoretically, it contributes new insight on the expressiveness of process calculi and the computational content of choice. More practically, it provides correctness arguments supporting the design of high-level concurrent languages on top of process calculi. Furthermore, it is tightly related to the distributed implementation of synchronization and selective communication [Mit86, PS92, Kna93, BG95].

Our interest in the study of choice encodings originates from the design and implementation of the high-level concurrent language Pict [PT95, PT99], an asynchronous choice-free π-calculus [HT91, Bou92] enriched with several layers of encoded syntactic sugar. The abstract machine of Pict does not provide instructions for selective communication; instead, choice is provided as a library module by a straightforward encoding. Surprisingly, however, this encoding turns out not to be valid with respect to standard weak bisimulation.

We study choice encodings in the π-calculus with *asynchronous messages* (or equivalently, with non-blocking output prefix). This setting has received increasing attention in recent years. In the ν-calculus, asynchrony was treated using a non-standard labeled semantics [HT91, HT92, Hon92]; the mini π-calculus used a chemical semantics [Bou92]; it has also been investigated using reduction semantics [HY95], concurrent combinators [HY94a, HY94b], and output-only barbed congruences [Ode95a, FG96]. Only recently, it has been extended with an input-guarded choice operator, equipped with a standard labeled semantics, and studied with bisimulation from an asynchronous observer's viewpoint [ACS98].

We use standard notations for restriction $(x)P$ of name x to process P, parallel composition $P_1|P_2$, input $y(x).P$ of a name from channel y for use as x in P, and output $\overline{y}z$ of name z on channel y. Furthermore, $\prod$ and $\sum$ denote indexed parallel composition and input-guarded choice, respectively. For convenience, we introduce the conditional form $\mathsf{if}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q$, which performs a case analysis driven by the special names t and f by reading from l either t or f and behaving afterwards like P or Q, respectively.

We study two variants of the choice encoding. The non-divergent version, which is more interesting from a pragmatic perspective, will occupy most of our attention. For each choice expression $\sum_{j\in J} y_j(x).P_j$, the translation

$$\mathcal{C}\left[\!\!\left[\ \sum_{j\in J} y_j(x).P_j\ \right]\!\!\right] \stackrel{\text{def}}{=} (l)\Big(\ \bar{l}\mathsf{t}\ \mid\ \prod_{j\in J} \mathit{Branch}_l\langle\ y_j(x).P_j\ \rangle\ \Big)$$

runs a mutual exclusion protocol, installing a local lock—a message that carries a special name—on the parallel composition of its branches. The branches

$$\mathit{Branch}_l\langle\ y_j(x).P_j\ \rangle \stackrel{\text{def}}{=} y_j(x)\ .\ l(b)\ .\ \mathsf{if}\ b\ \mathsf{then}\ (\ \mathcal{C}[\![\,P_j\,]\!] \mid \bar{l}\mathsf{f}\ )\ \mathsf{else}\ (\ \overline{y_j}x \mid \bar{l}\mathsf{f}\ )$$

concurrently try to (destructively) test the lock after reading messages from the environment. Only the first branch managing to interrogate the lock (via $l(b)$) will proceed with its continuation (then) and thereby commit the choice — every other branch will then be forced to resend its message and abort its continuation (else). The resending of messages by non-chosen branches essentially reflects the asynchronous character of the encoding. For an asynchronous observer, who can not detect when a message is consumed by a receptor, the resending of messages is immaterial, and so this encoding seems intuitively to be correct.

However, even for the asynchronous observer, it turns out that source terms and their $\mathcal{C}$-translations are not weakly bisimilar. The reason is that the latter carry out commitments only gradually, resulting in intermediate states that do not correspond to any source term. In order to deal with partially committed states, we instead characterize the correctness of the encoding as a pair of simulations which are *coupled* by requiring that less committed (i.e., simulating) processes can always internally evolve into more committed (i.e., simulated) processes [PS92].

For comparison, we also study another encoding that introduces an alternate path in each branch of a choice allowing it to "back out" and return to its initial state after it has been given the lock. This encoding avoids gradual commitments and can, in fact, be proven fully abstract with respect to weak bisimilarity. However, it is pragmatically unsatisfactory since it introduces divergence.

The remainder of the paper is organized as follows. We first introduce the setting of an asynchronous π-calculus (§2) and review some standard notions of correctness (§3). We then present the divergence-free encoding $\mathcal{C}$ and the divergent encoding $\mathcal{D}$, followed by an example that shows the failure of full abstraction of the $\mathcal{C}$-encoding with respect to weak bisimulation $\approx$ (§4). Using an intermediate language which lets us factor the $\mathcal{C}$-encoding into two steps, we define two *decoding* functions that constitute an asynchronous coupled simulation. This leads to our main result: for all source terms S,

$$S \rightleftarrows \mathcal{C}[\![S]\!]$$

where $\rightleftarrows$ is (asynchronous) coupled simulation equivalence; we also prove that the $\mathcal{C}$-encoding is divergence-free, and we sketch a proof that $S \approx \mathcal{D}[\![S]\!]$ (§5). Finally, we offer some concluding remarks and hint at related and future work (§6).

2 Technical preliminaries: The asynchronous π-calculus

Many variants of the π-calculus [MPW92] have appeared in the recent process algebra literature. We use here a version which is close to the core language of Pict [PT95, PT99]: an asynchronous, first-order, monadic π-calculus [HT91, Bou92], where replication is restricted to input processes and evaluated lazily [HY95, MP95], i.e., copies of a replicated input are spawned during communication.

2.1 Syntax

Let $\mathbf{N}$ be a countable set of *names*. Then, the set $\mathbb{P}$ of *processes* P is defined by

$$P \quad ::= \quad (x)P \quad | \quad P|P \quad | \quad \overline{y}z \quad | \quad \mathbf{0} \quad | \quad R \quad | \quad \mathbf{!}R$$
$$R \quad ::= \quad y(x).P$$

where $x, y, z \in \mathbf{N}$. The semantics of restriction, parallel composition, input, and output, is standard. The form $\mathbf{!}y(x).P$ is the replication operator restricted to input-prefixes. In $\overline{y}z$ and $y(x)$, the name y is called the *subject*, whereas x, z are called *objects*. We refer to outputs as *messages* and to (ordinary or replicated) inputs as *receptors*. A term is *guarded* when it occurs as a subterm of an input prefix R. As a notational abbreviation, if the object in some input or output is of no importance in some term, then it may be completely omitted, i.e., $y(x).P$ with $x \notin P$ and $\overline{y}z$ are written $y.P$ and $\overline{y}$. The usual constant $\mathbf{0}$ denoting the inactive process can be derived as $(x)\overline{x}$, but we prefer to include it explicitly for a clean specification of structural and operational semantics rules.

The definitions of name substitution and α-conversion are standard. A name x is *bound* in P, if P contains an x-binding operator, i.e., either a restriction $(x)P$ or an input prefix $y(x).P$ as a subterm. A name x is *free* in P if it occurs outside the scope of an x-binding operator. We write $\mathsf{bn}(P)$ and $\mathsf{fn}(P)$ for the sets of P's bound and free names; $\mathsf{n}(P)$ is their union. Renaming of bound names by α-conversion $=_\alpha$ is as usual. Substitution $P\{^z/_x\}$ is given by replacing all free occurrences of x in P with z, first α-converting P to avoid capture.

Operator precedence is, in decreasing order of binding strength: (1) prefixing, restriction, (2) substitution, (3) parallel composition.

2.2 Operational semantics

Let $y, z \in \mathbf{N}$ be arbitrary names. Then, the set $\mathbf{L}$ of *labels* μ is generated by

$$\mu \ ::= \ \overline{y}(z) \quad | \quad \overline{y}z \quad | \quad yz \quad | \quad \tau$$

representing bound and free output, early input, and internal action. The functions bn and fn yield the bound names, i.e., the objects in bound (bracketed) outputs, and free names (all others) of a label. We write $\mathsf{n}(\mu)$ for their union $\mathsf{bn}(\mu) \cup \mathsf{fn}(\mu)$.

The operational semantics for processes is given as a transition system with $\mathbb{P}$ as its set of states. The transition relation $\longrightarrow \subseteq \mathbb{P} \times \mathbf{L} \times \mathbb{P}$ is defined as the smallest relation generated by the set of rules in Table 1. We use an *early* instantiation scheme as expressed in the *INP* and *COM*/*CLOSE*-rules, since it allows us to define bisimulation without clauses for name instantiation and since it allows for a more intuitive modeling in §4, but this decision does not affect the validity of our results. Rule *OPEN* prepares for scope extrusion, whereas in *CLOSE* the previously opened scope of a bound name is closed upon its reception.

OUT: $\quad \overline{y}z \xrightarrow{\overline{y}z} \mathbf{0}$

INP: $\quad y(x).P \xrightarrow{yz} P\{^z/_x\}$

R-INP: $\quad !y(x).P \xrightarrow{yz} P\{^z/_x\} \mid !y(x).P$

COM$_1^*$: $$\frac{P_1 \xrightarrow{\overline{y}z} P_1' \qquad P_2 \xrightarrow{yz} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

OPEN: $$\frac{P \xrightarrow{\overline{y}z} P'}{(z)P \xrightarrow{\overline{y}(z)} P'} \qquad \textit{if } y \neq z$$

CLOSE$_1^*$: $$\frac{P_1 \xrightarrow{\overline{y}(z)} P_1' \qquad P_2 \xrightarrow{yz} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} (z)(\, P_1' \mid P_2' \,)} \qquad \textit{if } z \notin \mathsf{fn}(P_2)$$

PAR$_1^*$: $$\frac{P_1 \xrightarrow{\mu} P_1'}{P_1 \mid P_2 \xrightarrow{\mu} P_1' \mid P_2} \qquad \textit{if } \mathsf{bn}(\mu) \cap \mathsf{fn}(P_2) = \emptyset$$

RES: $$\frac{P \xrightarrow{\mu} P'}{(x)P \xrightarrow{\mu} (x)P'} \qquad \textit{if } x \notin \mathsf{n}(\mu)$$

ALPHA: $$\frac{\hat{P} \xrightarrow{\mu} \hat{P}'}{P \xrightarrow{\mu} \hat{P}'} \qquad \textit{if } P =_\alpha \hat{P}$$

**: and the evident symmetric rules COM$_2$, CLOSE$_2$, and PAR$_2$*

Table 1: Transition semantics

Weak arrows ($\Rightarrow$) denote the reflexive and transitive closure of internal transitions; arrows with hats denote that two processes are either related by a particular transition or else equal in the case of an internal transition.

$$\Rightarrow \stackrel{\text{def}}{=} \xrightarrow{\tau}{}^{*} \qquad \xrightarrow{\widehat{\mu}} \stackrel{\text{def}}{=} \begin{cases} \xrightarrow{\mu} & \textit{if } \mu \neq \tau \\ \xrightarrow{\tau} \cup = & \textit{if } \mu = \tau \end{cases} \qquad \begin{array}{l} \stackrel{\widehat{\mu}}{\Longrightarrow} \stackrel{\text{def}}{=} \Rightarrow \xrightarrow{\widehat{\mu}} \Rightarrow \\ \stackrel{\mu}{\Longrightarrow} \stackrel{\text{def}}{=} \Rightarrow \xrightarrow{\mu} \Rightarrow \end{array}$$

2.3 Bisimulation

Two process systems are equivalent when they allow us to *observe* the same operational behavior. Bisimulation equivalence is defined as the mutual simulation of single computation steps resulting in equivalent system states. In the standard literature on bisimulations, e.g. [Mil89, MPW92], a *simulation* is a binary relation $\mathcal{S}$ on processes such that $(P, Q) \in \mathcal{S}$ implies, for arbitrary label μ:

- if $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{S}$.

A *bisimulation* is a simulation whose opposite is again a simulation. In this subsection, we review various refinements of standard bisimulation: we define its asynchronous variant, identify a few structural laws, refine it to a preorder that takes efficiency into account, and finally refine it to deal with divergence.

Asynchrony

In calculi with synchronous message-passing, output- and input-transitions are dealt with symmetrically in the definition of bisimulation. In contrast, the concept of asynchronous messages suggests a non-standard way of observing processes. Since the sending of a message to an observed system is not blocking for the observer, the latter can not immediately detect whether the message was actually received, or not. The only possible observations are messages eventually coming back from the system. Different formulations of asynchronous bisimulation (all inducing the same equivalence) have been proposed in the literature, based on a modified labeled input rule [HT92], on output-only barbed congruences [HY95], and on a standard labeled semantics with asynchronous observers [ACS98]. Here, we follow the latter approach. Unless otherwise stated, we implicitly assume an asynchronous interpretation of observation throughout the paper.

Definition 2.3.1 (Simulation, bisimulation) *A binary relation $\mathcal{S}$ on processes is a* strong simulation *if $(P, Q) \in \mathcal{S}$ implies:*

- *if $P \xrightarrow{\mu} P'$, where μ is either τ or output with $\mathsf{bn}(\mu) \cap \mathsf{fn}(P|Q) = \emptyset$, then there is Q' such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{S}$*

α-conversion	P	$\equiv$	Q	*if*	$P =_\alpha Q$
associativity	$P \mid (Q \vert R)$	$\equiv$	$(P \vert Q) \mid R$		
commutativity	$P \mid Q$	$\equiv$	$Q \mid P$		
neutrality	$P \mid \mathbf{0}$	$\equiv$	P		
scope extrusion	$(y)P \mid Q$	$\equiv$	$(y)(P \vert Q)$	*if*	$y \notin \mathsf{fn}(Q)$
scope elimination	$(y)Q$	$\equiv$	Q	*if*	$y \notin \mathsf{fn}(Q)$

Table 2: Structural laws

- $(\ \overline{a}z|P\ ,\ \overline{a}z|Q\ ) \in \mathcal{S}$ *for arbitrary messages* $\overline{a}z$.

$\mathcal{B}$ *is called a* strong bisimulation *if both* $\mathcal{B}$ *and* $\mathcal{B}^{-1}$ *are strong simulations. Two processes* P *and* Q *are* strongly bisimilar*, written* $P \sim Q$*, if they are related by some strong bisimulation.*

Replacing $Q \xrightarrow{\mu} Q'$ *with* $Q \xRightarrow{\widehat{\mu}} Q'$ *in this definition yields the* weak *versions of the corresponding simulations. Write* $\approx$ *for weak bisimulation. Process* Q weakly simulates P*, written* $P \preccurlyeq Q$*, if there is a weak simulation* $\mathcal{S}$ *with* $(P, Q) \in \mathcal{S}$*.*

Fact 2.3.2 $\approx$ *is a congruence on* $\mathbb{P}$*.* [Hon92, ACS98]

Structural laws

Certain laws on processes have been recognized as having merely "structural" content; they are valid with respect to all different kinds of behavioral congruences, equivalences and preorders, including strong bisimulation (the finest "reasonable" equivalence). In this paper, we use a few structural laws (indicated by the symbol $\equiv$) listed in Table 2 in order to simplify the presentation of some derivation sequences of transitions in the proofs of Section 5.

Fact 2.3.3 $\quad \equiv\ \subset\ \sim$.

In particular, we work *up to* α-conversion, where appropriate, i.e., we omit to mention the implicit application of rule *ALPHA* since it may be captured by a simple structural transformation. Thus, we silently identify processes or actions which only differ in the choice of bound names. Furthermore, due to the associativity law for composition, we omit brackets in multiple parallel composition and use finite parallel composition $\prod$ with the usual meaning.

Efficiency

Often, weakly bisimilar processes differ only in the number of internal steps. The *expansion preorder* [AH92] takes this into account by stating that one process engages in at least as many internal actions as another. We use expansions to define an up-to technique in Section 2.5, and to precisely formulate the correctness of an encoding in Section 5.5.

Deviating from the standard presentation of expansion, we introduce some auxiliary terminology that will be useful in the next subsection for capturing aspects of divergence. It is partly inspired by the notion of progressing bisimulation in CCS [MS92b].

Definition 2.3.4 *A weak simulation $\mathcal{S}$ is called*

- progressing*, if* $P \xrightarrow{\mu} P'$ *implies that there is* Q' *with* $Q \stackrel{\mu}{\Longrightarrow} Q'$ *such that* $(P', Q') \in \mathcal{S}$
- strict*, if* $P \xrightarrow{\mu} P'$ *implies that there is* Q' *with* $Q \xrightarrow{\widehat{\mu}} Q'$ *such that* $(P', Q') \in \mathcal{S}$

for all $(P, Q) \in \mathcal{S}$ *and for all* μ *being* τ *or output with* $\mathsf{bn}(\mu) \cap \mathsf{fn}(P|Q) = \emptyset$.

Note that a weak simulation is strong if it is both progressing and strict.

Definition 2.3.5 (Expansion) *A binary relation $\mathcal{E}$ on processes is an* expansion *if $\mathcal{E}$ is a progressing simulation and $\mathcal{E}^{-1}$ is a strict simulation. Process Q* expands *P, written $P \lesssim Q$, if there is an expansion $\mathcal{E}$ with $(P, Q) \in \mathcal{E}$.*

Fact 2.3.6 $\sim \;\subset\; \lesssim \;\subset\; \approx$.

Divergence

Since we are going to formally reason about divergence properties of encodings in Section 5.8, we need some basic definitions. According with standard intuitions, a process P is said to be *divergent*, written $P{\Uparrow}$, if there is an infinite sequence of τ-steps starting at P; otherwise it is called *convergent*, written $P{\Downarrow}$.

Weak (bi-)simulation is insensitive to the divergence of processes. This lack of expressiveness arises from the definition of weak simulation, which may always choose to mimic τ-steps trivially. Weak bisimulation, or observation equivalence, may therefore equate two processes exactly one of which is diverging: it simply ignores the existence of infinite τ-sequences. Enhancements of bisimulation have been investigated that take divergence behavior explicitly into account, resulting in preorders among bisimilar processes [Wal90].

For our purposes in Section 5.8, the simpler property of *preserving divergence* will suffice. A weak simulation $\mathcal{S}$ has this property if $P{\Uparrow}$ implies $Q{\Uparrow}$ for all $(P, Q) \in \mathcal{S}$. Intuitively, when required to weakly simulate an infinite τ-sequence, $\mathcal{S}$ must *progress* infinitely often. Let us introduce some further notation (inspired by Priese [Pri78]) to make precise what this means.

Let $\xrightarrow{\tau}^n$ denote a τ-sequence of length n and $\xrightarrow{\tau}^+ \stackrel{\text{def}}{=} \xrightarrow{\tau}^n$ for some $n > 0$ denote a non-empty, but arbitrarily long, finite sequence of τ-steps.

Definition 2.3.7 (Eventually progressing simulation) *A weak simulation $\mathcal{S}$ is called* eventually progressing *if, for all $(P, Q) \in \mathcal{S}$, there is a natural number $k_P \in \mathbb{N}$ such that $P \xrightarrow{\tau}^n P'$ with $n > k_P$ implies that there is Q' with $Q \xrightarrow{\tau}^+ Q'$ such that $(P', Q') \in \mathcal{S}$.*

According to the definition, a simulation $\mathcal{S}$ is eventually progressing if, for sufficiently long finite τ-sequences, $\mathcal{S}$ must eventually reproduce a τ-step. In this respect, k_P is to be understood as an upper bound for the number of τ-steps starting from P that may be trivially simulated. Note that every progressing simulation is, by definition, eventually progressing with upper bound 0.

With a progressing simulation, every infinite sequence may be simulated by subsequently simulating sufficiently long finite subsequences non-trivially, i.e., such that they represent progress. This resembles the *chunk-by-chunk* idea of simulation due to Gammelgaard [Gam91].

Lemma 2.3.8 *Eventually progressing simulations preserve divergence.*

Proof: Let $\mathcal{S}$ be a progressing simulation and $(P, Q) \in \mathcal{S}$. If $P\Uparrow$ then there is $P \xrightarrow{\tau}^\omega$. Since $\mathcal{S}$ is progressing, there is $k_P \in \mathbb{N}$ such that $P \xrightarrow{\tau}^{k_P+1} P' \xrightarrow{\tau}^\omega$ and $Q \xrightarrow{\tau}^+ Q'$ with $(P', Q') \in \mathcal{S}$. Since now $P'\Uparrow$, we can repeat the procedure infinitely often. $\square$

2.4 When weak bisimulation is too strong ...

Every bisimulation $\mathcal{B}$ can be regarded as a pair $(\mathcal{S}_1, \mathcal{S}_2)$ of contrary simulations $\mathcal{S}_1$ and $\mathcal{S}_2^{-1}$, where $\mathcal{S}_1$ and $\mathcal{S}_2$ contain exactly the same pairs of processes, i.e., $\mathcal{S}_1 = \mathcal{B} = \mathcal{S}_2$. For some applications, this requirement is too strong. For example, consider the following $\mathbb{P}$-processes:

$$
\begin{array}{lclccccc}
P & \stackrel{\text{def}}{=} & (i) & (& \overline{i} & \mid i.A & \mid i.B & \mid i.C \ ) \\
Q & \stackrel{\text{def}}{=} & (i_1)(i_2) & (& \overline{i_1} \mid \overline{i_2} & \mid i_1.A & \mid i_1.(\, i_2.B \mid i_2.C\,) &)
\end{array}
$$

where $i, i_1, i_2 \notin \mathsf{fn}(A, B, C)$. The example was originally introduced in CCS, using choice operators [PS92]. In fact, both P and Q implement two different versions of some behavior that performs an internal choice among the processes A, B, and C by the concurrent race of inputs for an output on some shared internal channel. The difference is that P only uses one internal channel (i), whereas Q uses two of them (i_1 and i_2) and, as a result, needs two internal steps in order to decide which of B or C is chosen in the case that A was preempted. Hence, the choice implemented by P is atomic, whereas the choice of Q is gradual.

Although intuitively P and Q might be regarded as observably equivalent by disregarding the internal choices, which are present in Q but not in P, they are not weakly bisimilar. According to the derivation trees up to $\sim$ (using the law $(i)(\, i.P \mid Q\,) \sim Q$, if $i \notin \mathsf{fn}(Q)$) and with $BC := (i_2)(\, \overline{i_2} \mid i_2.B \mid i_2.C\,)$,

$$\begin{tikzcd}
& P \arrow[dl, "\tau"'] \arrow[d, "\tau"'] \arrow[dr, "\tau"] & \\
A & B & C
\end{tikzcd}
\qquad
\begin{tikzcd}
& Q \arrow[dl, "\tau"'] \arrow[dr, "\tau"] & & \\
A & & BC \arrow[dl, "\tau"'] \arrow[dr, "\tau"] & \\
& B & & C
\end{tikzcd}$$

the state BC cannot be related to any of the states beneath P, which would both simulate BC and be simulated by BC. At best, we can find two contrary simulations $\mathcal{S}_1$ and $\mathcal{S}_2^{-1}$ with

$$\begin{array}{rcl}
\mathcal{S} & \stackrel{\text{def}}{=} & \{\, (P,Q),\ (A,A),\ (B,B),\ (C,C),\ \ldots\,\} \\
\mathcal{S}_1 & \stackrel{\text{def}}{=} & \mathcal{S} \cup \{\, (B,\ BC),\ (C,\ BC)\,\} \\
\mathcal{S}_2 & \stackrel{\text{def}}{=} & \mathcal{S} \cup \{\, (P,\ BC)\,\}
\end{array}$$

which, unfortunately, do not coincide. The distinguishing pairs express the problem with the *partially committed* τ-derivative BC of Q: on the one hand, it cannot be simulated by any τ-derivative of P due to their lack of ability to reach both B and C; on the other hand, it can itself no longer simulate P, because it has lost the ability to reach A.

As an appropriate mathematical tool to handle situations as the above example, Parrow and Sjödin developed the notion of *coupled simulation* [PS92]: two contrary simulations are no longer required to coincide, but only to be coupled in a certain way. Several candidates have been presented for what it means to be coupled. No coupling at all would just lead to the notion of mutual simulation. A non-trivial notion of coupling was based on the property of *stability* by requiring the coincidence of two contrary simulations in at least the stable states. This style induces a relation which is an equivalence only for convergent processes, and it has been proven to be strictly weaker than bisimulation and strictly stronger than testing equivalence [PS92]; indeed, the two processes P and Q of the introductory example are equivalent in that sense, as formalized in Definition 2.4.1 below.

In this paper, we use a generalization for divergent processes, as suggested by van Glabbeek [Gla93, PS94], where coupling requires the ability of a simulating process to evolve into a simulated process by internal action. We recapitulate the formal definition:

Definition 2.4.1 (Coupled simulation) *A* mutual simulation *is a pair* $(\mathcal{S}_1, \mathcal{S}_2)$*, where* $\mathcal{S}_1$ *and* $\mathcal{S}_2^{-1}$ *are weak simulations. A* coupled simulation *is a mutual simulation* $(\mathcal{S}_1, \mathcal{S}_2)$ *satisfying*

- *if $(P, Q) \in \mathcal{S}_1$, then there is some Q' such that $Q \Rightarrow Q'$ and $(P, Q') \in \mathcal{S}_2$;*
- *if $(P, Q') \in \mathcal{S}_2$, then there is some P' such that $P \Rightarrow P'$ and $(P', Q') \in \mathcal{S}_1$.*

Processes P and Q are coupled simulation equivalent *(or* coupled similar*), written $P \rightleftarrows Q$, if they are related by both components of some coupled simulation.*

Using dotted lines to represent the simulations, the coupling property of $(\mathcal{S}_1, \mathcal{S}_2)$ may be depicted as an 'internally out-of-step bisimulation' by:

$$\begin{tikzcd}
 & P \arrow[rr, Rightarrow] \arrow[dl, dash, dotted, "\mathcal{S}_1"'] \arrow[dr, dash, dotted, "\mathcal{S}_2"] & & P' \arrow[dl, dash, dotted, "\mathcal{S}_1"] & \cdots \\
Q \arrow[rr, Rightarrow] & & Q' & \cdots &
\end{tikzcd}
\quad \text{or:} \quad
\begin{tikzcd}
 & P \arrow[rr, Rightarrow] \arrow[dl, dash, dotted, "\succcurlyeq"'] \arrow[dr, dash, dotted, "\succcurlyeq"] & & P' \arrow[dl, dash, dotted, "\succcurlyeq"] & \cdots \\
Q \arrow[rr, Rightarrow] & & Q' & \cdots &
\end{tikzcd}$$

Of two processes contained in one component relation of some coupled simulation, the simulated (more committed) process is always a bit ahead of its simulating (less committed) counterpart. Intuitively, 'Q coupled simulates P' means that 'Q is *at most as committed* as P' with respect to internal choices and that Q may internally evolve to a state Q' where it is *at least as committed* as P, i.e., where P coupled simulates Q'.

Fact 2.4.2 *Let $(\mathcal{S}_1, \mathcal{S}_2)$ be a coupled simulation. Then $(\mathcal{S}_2^{-1}, \mathcal{S}_1^{-1})$ is a coupled simulation.*

Observe that the pair $(\approx, \approx)$ is a coupled simulation by trivial coupling sequences, as motivated at the beginning of the section. Furthermore, the motivating example witnesses that coupled simulation equivalence is strictly coarser than weak bisimilarity.

Fact 2.4.3 $\approx \, \subset \, \rightleftarrows$.

On processes without infinite τ-sequences, coupled simulation is strictly finer than testing equivalence, so it represents a reasonable (and coinductively defined) candidate in the lattice of process equivalences [Gla93]. In general, it is compatible with all operators of $\mathbb{P}$.

Proposition 2.4.4 *$\rightleftarrows$ is a congruence on $\mathbb{P}$.*

Proof: Reflexivity is immediate by definition.

For symmetry, let $P_1 \rightleftarrows P_2$ by $(\mathcal{S}_1, \mathcal{S}_2)$ with $(P_1, P_2) \in \mathcal{S}_1 \cap \mathcal{S}_2$. This is equivalent to $(P_2, P_1) \in \mathcal{S}_2^{-1} \cap \mathcal{S}_1^{-1}$, and, by Fact 2.4.2, we have $P_2 \rightleftarrows P_1$.

For transitivity, let $P \rightleftarrows Q \rightleftarrows R$ due to their containment in coupled simulations $(P, Q) \in (\mathcal{S}_{PQ}, \mathcal{S}_{QP})$ and $(Q, R) \in (\mathcal{S}_{QR}, \mathcal{S}_{RQ})$. Now, let $\mathcal{S}_{PQR} := \mathcal{S}_{PQ}\mathcal{S}_{QR}$ and $\mathcal{S}_{RQP} := \mathcal{S}_{QP}\mathcal{S}_{RQ}$. Then, both $\mathcal{S}_{PQR}$ and $\mathcal{S}_{RQP}^{-1}$ are simulations by transitivity of simulation. For the coupling between $\mathcal{S}_{PQR}$ and $\mathcal{S}_{RQP}$, we show only one direction. Since $(\mathcal{S}_{PQ}, \mathcal{S}_{QP})$ is a coupled simulation, we know that $Q \Rightarrow Q'$ with $(P, Q') \in \mathcal{S}_{QP}$. Since $(Q, R) \in \mathcal{S}_{QR}$, this sequence can be simulated by $R \Rightarrow R'$ with $(Q', R') \in \mathcal{S}_{QR}$. Now, since $(\mathcal{S}_{QR}, \mathcal{S}_{RQ})$ is a coupled simulation, we have $R' \Rightarrow R''$ such that $(Q', R'') \in \mathcal{S}_{RQ}$. Therefore, we conclude that $R \Rightarrow R''$ with $(P, R'') \in \mathcal{S}_{QP}\mathcal{S}_{RQ} = \mathcal{S}_{RQP}$.

For congruence, we show that $\rightleftarrows$ is preserved by all operators in $\mathbb{P}$. For each pair (P_1, P_2) in $\rightleftarrows$, there is a witnessing coupled simulation $(\mathcal{S}_1, \mathcal{S}_2)$. Congruence of $\rightleftarrows$ requires the preservation of $\rightleftarrows$ under all operators in $\mathbb{P}$. It is known that weak simulation is preserved by all operators of $\mathbb{P}$. In addition, we only have to check that the required coupling is preserved correspondingly. This, however, is straightforward since the coupling is just the existence of some possibly trivial internal sequence. For parallel composition and restriction, the coupling is inherited directly from the components; for (replicated) guards, the coupling is trivial. $\square$

2.5 Up-to techniques

By the coinductive definition of bisimulation, a proof that two processes P and Q are bisimilar, i.e., $P \approx Q$, rests on the construction of *some* bisimulation $\mathcal{B}$ which contains the pair (P, Q). Up-to techniques have been introduced in order to improve the bisimulation proof technique by relaxing the proof obligations and thereby reducing the size of the witness relation $\mathcal{B}$ [Mil89, San95b]. We explain the idea by the notion of weak simulation up to expansion. (The composition of relations is denoted by juxtaposition.)

Definition 2.5.1 (Simulation up to expansion) *A binary relation $\mathcal{S}$ on processes is a* (weak) simulation up to $\lesssim$ *if $(P, Q) \in \mathcal{S}$ implies:*

- *if $P \xrightarrow{\mu} P'$, where μ is τ or an output with $\mathsf{bn}(\mu) \cap \mathsf{fn}(P|Q) = \emptyset$, then there is Q' such that $Q \xRightarrow{\hat{\mu}} Q'$ and $(P', Q') \in \mathcal{S} \lesssim$*
- *$(\, \overline{a}z|P \,,\, \overline{a}z|Q \,) \in \mathcal{S}$ for arbitrary messages $\overline{a}z$.*

Note that the first obligation on $\mathcal{S}$ does not require of Q' that $(P', Q') \in \mathcal{S}$, but only that Q' expands some process Q'' with $(P', Q'') \in \mathcal{S}$. The following lemma provides a proof technique for weak simulation based on the above definition.

Lemma 2.5.2 *If Q simulates P up to expansion, then $P \preccurlyeq Q$.*

Proof: Let $\mathcal{U}$ be a weak simulation up to expansion that contains (P, Q). Then the relation $\mathcal{U} \cup \{\, (P', Q') \mid \exists (P_0, Q_0) \in \mathcal{U}.\ \exists (P', Q'') \in \mathcal{U}.\ \exists \mu \in \mathbf{L}.\ (P \xrightarrow{\mu} P' \wedge Q \xRightarrow{\hat{\mu}} Q'' \lesssim Q') \,\}$ is a weak simulation and contains (P, Q). $\square$

The following lemma allows us to compose coupled simulations with bisimulations.

Lemma 2.5.3 *Let $(\mathcal{S}_1, \mathcal{S}_2)$ be a coupled simulation and $\mathcal{B}$ a weak bisimulation. Then the composite pair $(\mathcal{S}_1\mathcal{B}, \mathcal{S}_2\mathcal{B})$ is again a coupled simulation.*

Proof: We have to prove that both $\mathcal{S}_1\mathcal{B}$ and $(\mathcal{S}_2\mathcal{B})^{-1}$ are weak simulations and that there is the desired coupling via internal transition sequences. All of these facts are straightforward. We only show the case for the reachability of coupled states.

Let $(P, R) \in \mathcal{S}_1\mathcal{B}$ via Q, i.e., $(P, Q) \in \mathcal{S}_1$ and $(Q, R) \in \mathcal{B}$. Then, since $(\mathcal{S}_1, \mathcal{S}_2)$ is a coupled simulation, we know that there is Q' with $Q \Rightarrow Q'$ and $(P, Q') \in \mathcal{S}_2$. Furthermore, from $(Q, R) \in \mathcal{B}$ we know that there is some R' with $R \Rightarrow R'$ and $(Q', R') \in \mathcal{B}$. Thus, $(P, R') \in \mathcal{S}_2\mathcal{B}$.

The proof of the second clause for coupling is even simpler. Let $(P, R) \in \mathcal{S}_2\mathcal{B}$ via Q, i.e., $(P, Q) \in \mathcal{S}_2$ and $(Q, R) \in \mathcal{B}$. From $(\mathcal{S}_1, \mathcal{S}_2)$ being a coupled simulation we know that there is P' with $P \Rightarrow P'$ and $(P', Q) \in \mathcal{S}_2$. Then, immediately $(P', R) \in \mathcal{S}_1\mathcal{B}$ via Q. $\square$

Thus, in order to prove that two processes S and T are coupled similar, it suffices to show that S is coupled similar to some other process A (this might be considerably easier), which, in turn, is bisimilar to the process T. We may call the composite $(\mathcal{S}_1\mathcal{B}, \mathcal{S}_2\mathcal{B})$ a *coupled simulation up to bisimulation*, although this is not exactly in the spirit of up-to techniques. There, the aim is to reduce the size of relations that are necessary to prove that two processes are related. Here, we do not decrease the size, but simply carry out the proof on another, but bisimilar, set of terms that may provide richer structure for actually doing the proof.

3 Discussion: Correctness of encodings

In this section, we briefly digress to review a few known notions of correctness and discuss their advantages and disadvantages. Thereby, we aim at precisely characterizing the class of *non-prompt* encodings that is represented by the choice encodings of the Introduction and Section 4. As shown later on in Section 5, it is the non-prompt character of choice encodings that complicates the definition of and reasoning about their correctness.

Intuitively, if we can compare terms and their translations directly, then we may require that every source term S and its translation $[\![S]\!]$ should be **semantically equivalent**

$$S \asymp [\![S]\!]$$

where $\asymp$ denotes some notion of equivalence; we provide examples with Theorems 5.6.1 and 5.7.4. The stronger the employed equivalence, the more we are tempted to accept $[\![\ ]\!]$ as being correct. For process calculi, some prominent candidates among the vast number of equivalences are (with decreasing ability to distinguish process terms): strong and weak bisimulation, testing, and trace equivalence. Since most encodings introduce additional computation steps compared to the behavior of source terms, we may hardly expect correctness up to strong bisimulation. Weak bisimulation may be applicable whenever the additional steps are internal. Furthermore, bisimulation comes with coinductive proof techniques. Testing equivalences that are strictly weaker than bisimulation may often be sufficient as correctness criteria, but they lack a convenient proof technique. Therefore, bisimulation is also appealing in those cases where, for example, some testing equivalence would suffice.

In general, however, we cannot assume that we have a formal setting at hand that allows us to compare terms and their translations directly. The notion of **full abstraction** has been developed to get around this problem. Here, correctness is expressed as the preservation and reflection of equivalence of source terms. Let $\asymp_{\mathsf{s}}$ and $\asymp_{\mathsf{t}}$ denote equivalences of the source and the target language, respectively. Then, the full abstraction property is formulated as:

$$S_1 \asymp_{\mathsf{s}} S_2 \text{ if and only if } [\![\, S_1 \,]\!] \asymp_{\mathsf{t}} [\![\, S_2 \,]\!].$$

Up to the chosen notions of equivalence, fully abstract encodings allow us—for reasoning about terms—to freely switch between the source and target languages in both directions. Note that, usually, the reflection (*if*) of equivalence, often called *adequacy*, is relatively easy to establish. In contrast, to prove the preservation (*only if*) of equivalence is not an easy task: the chosen notions of equivalence should be insensitive to the additional computation steps that are introduced by the encoding; moreover, when one is interested in congruences, the equivalence has to be preserved in not only translated high-level contexts, but arbitrary low-level contexts. Yet, only when both reflection and preservation hold the theory and proof techniques of a low-level language can always be used for reasoning about high-level terms; preservation then provides behavioral completeness, reflection provides behavioral soundness.

Often, e.g. for encodings of object-oriented languages, the source language is not *a priori* equipped with a notion of equivalence. Thus, we may not be able to check the encoding's correctness via a full abstraction result. The notion of **operational correspondence** was therefore designed to capture correctness as the preservation and reflection of execution steps as defined by an operational semantics of the source and the target languages, and expressed in the model of transition systems which specify the execution of terms. Let $\rightarrow_{\mathsf{s}}$ and $\rightarrow_{\mathsf{t}}$ denote transition relations on the source and target language, respectively, and let $\Rightarrow_{\mathsf{s}}$ and $\Rightarrow_{\mathsf{t}}$ denote their reflexive transitive closure. Then, operational correspondence is characterized by two complementary propositions, which we briefly call *completeness* ($\mathfrak{C}$) and *soundness* ($\mathfrak{S}$).

Completeness (Preservation of execution steps.) The property

$$\text{if } S \rightarrow_{\mathsf{s}} S',\ \text{then } [\![\, S \,]\!] \Rightarrow_{\mathsf{t}} [\![\, S' \,]\!] \tag{$\mathfrak{C}$}$$

states that all possible executions of S may be simulated by its translation, which is naturally desirable for most encodings.

Soundness (Reflection of execution steps.) The converse of completeness, i.e., the property

$$\text{if } [\![\, S \,]\!] \Rightarrow_{\mathsf{t}} [\![\, S' \,]\!] \text{ then } S \Rightarrow_{\mathsf{s}} S',$$

is, in general, not strong enough since it deals neither with all possible executions of translations nor with the behavior of intermediate states between $[\![\, S \,]\!]$ and $[\![\, S' \,]\!]$. For example, nondeterministic or divergent executions, sometimes regarded as undesirable, could although starting from a translation $[\![\, S \,]\!]$ never again reach a state that is a translation $[\![\, S' \,]\!]$. A refined property may consider the behavior of intermediate states to some extent:

$$\text{if } [\![\, S \,]\!] \rightarrow_{\mathsf{t}} T \text{ then there is } S \rightarrow_{\mathsf{s}} S' \text{ such that } T \asymp_{\mathsf{t}} [\![\, S' \,]\!] \tag{$\mathfrak{I}$}$$

says that initial steps of a translation can be simulated by the source term such that the target-level derivative is equivalent to the translation of the source-level derivative.

Let us call a target-level step *committing* if it directly corresponds to some source-level step. It should be clear that only *prompt* encodings, i.e., those where initial steps of literal translations are committing, will satisfy $\mathfrak{I}$. As a matter of fact, most encodings studied up to now in the literature are prompt. Promptness also leads to 'nice' proof obligations since it requires case analysis over single computation steps.

However, non-prompt encodings do not satisfy $\mathfrak{I}$; like choice encodings, they allow administrative (or book-keeping) steps to *precede* a committing step. Sometimes (cf. [Ama94]), these administrative steps are well behaved in that they can be captured by a confluent and strongly normalizing reduction relation. Then, the encoding is optimized to perform itself the initial administrative overhead by mapping source terms onto administrative normal forms to satisfy $\mathfrak{I}$. A satisfyingly general approach to take administrative steps into account is:

$$\text{if } [\![\, S \,]\!] \Rightarrow_{\mathsf{t}} T \text{ then there is } S \Rightarrow_{\mathsf{s}} S' \text{ such that } T \Rightarrow_{\mathsf{t}} [\![\, S' \,]\!] \tag{$\mathfrak{S}$}$$

says that arbitrary sequences of target steps are simulated (up to completion) by the source term. It takes all derivatives T—including intermediate states—into account and does not depend on the encoding being prompt or normalizable. Thus, $\mathfrak{S}$ is rather appealing. However, it only states correspondence between sequences of transitions and is therefore, in general, rather hard to prove, since it involves analyzing arbitrarily long transition sequences between $[\![\, S \,]\!]$ and T (see [Wal95] for a successful proof).

Finally, note that a proof that source terms and their translations are the same up to some operationally defined notion of equivalence gives full abstraction up to that equivalence and operational correspondence for free (see Corollaries 5.6.4 and 5.7.6, and the discussion at the end of Section 5.4).

4 Encoding input-guarded choice, asynchronously

This section defines encodings of the asynchronous π-calculus with input-guarded choice $\mathbb{P}^{\Sigma}$ into its choice-free fragment $\mathbb{P}$ (§4.1): a divergence-free choice encoding, and a divergent variant of it (§4.2). The essential idea of the encodings is that a branch may consume a message before it checks whether it was allowed to do so; if yes, then it may proceed, otherwise it simply resends the consumed message. Such protocols are only correct with respect to asynchronous observation, which is insensitive to the temporary buffering of messages. An example process term (§4.3) exhibits the essential difference between the two choice encodings and prepares the ground for a discussion of possible correctness statements.

In the above-mentioned protocols, mutual exclusion among the branches of a choice is implemented by a concurrent race for a shared lock. Since the lock may be most succinctly expressed by means of some form of boolean values, we introduce (in §4.1) an intermediate language that provides a convenient primitive operator for testing boolean-valued messages; §4.4 gives a careful treatment of the expansion of those intermediate terms.

4.1 The setting

An encoding is a function from some source syntax into some target syntax. This subsection introduces the language $\mathbb{P}^{\Sigma}$ that represents the source syntax. In addition, we refine the setting by an intermediate language $\mathbb{P}^{\mathsf{test}}$ to be used instead of the intended target language $\mathbb{P}$.

$$\begin{tikzcd}
\mathbb{S} := & \mathbb{P}^{\Sigma} \arrow[d, dotted] \arrow[dr, dotted] & & \\
& \mathbb{P} & \mathbb{P}^{\mathsf{test}} \arrow[l, dotted] & =: \mathbb{T}
\end{tikzcd}$$

In diagrams like the above, dotted arrows are just placeholders for encoding functions.

Source language We introduce choice as a finitely indexed operator on input terms. Its behavior is specified by the operational semantics rule in Table 3, which formally describes that each branch in a choice may be selected and consume an external message, preempting all of its competing branches. The set $\mathbb{P}^{\Sigma}$ of processes with *input-guarded choice* is generated by adding a clause for $\sum$-expressions to the grammar of $\mathbb{P}$:

$$P \ ::= \ \ldots \quad \Big| \quad \sum_{j \in J} R_j$$

where J is some finite indexing set. We also use the abbreviation $R_1 + R_2$ to denote binary input-guarded choice. The labeled transition semantics of $\mathbb{P}^{\Sigma}$ is the relation generated by the rules in Table 1 and rule *C-INP* in Table 3.

Target language(s) Instead of directly defining the encodings from $\mathbb{P}^{\Sigma}$ into $\mathbb{P}$, we use an intermediate language $\mathbb{P}^{\mathsf{test}}$ that provides special *boolean* names, and also a conditional form $\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q$ for testing the (boolean) value of messages along some channel l.

Let $\mathbf{B} := \{\mathsf{f}, \mathsf{t}\}$ be the set of *special names* that we interpret as boolean values. Let $\mathbf{V} := \mathbf{N} \uplus \mathbf{B}$ be referred to as the set of *values*. Then the set $\mathbb{P}^{\mathsf{test}}$ of processes with *conditionals* is defined by adding a clause for test-expressions to the grammar of $\mathbb{P}$:

$$P \ ::= \ \ldots \quad \Big| \quad \mathsf{test}\ y\ \mathsf{then}\ P\ \mathsf{else}\ P$$

In addition, in order to forbid restriction on, communication on, and substitution for booleans, we adopt some naming conventions concerning their admissible occurrences in the clauses of the grammar of $\mathbb{P}$ (cf. §2) by requiring that $x, y \in \mathbf{N}$ and $z \in \mathbf{V}$. Thus, the only use of booleans in $\mathbb{P}^{\mathsf{test}}$ is as objects in messages. Note that test-expressions can be regarded as abbreviations of if-expressions that are prefixed with some input of a boolean message.

$$\mathsf{test}\ y\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \quad \widehat{=} \quad y(b)\ .\ \mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$$

The reason for using test instead of if is, on the one hand, better readability and, on the other hand, that if, like matching operators, destroys some congruence properties of the language. In order to model the behavior of test-expressions that can interact with messages, we supply an operational semantics by the rules in Table 4 that properly fits with the labeled transition system semantics of $\mathbb{P}$ and the explanation in terms of if.[1] The

[1] By using an *early* instantiation scheme, the interaction between boolean messages and test-expressions is handled by the standard *COM*-rule; otherwise, we would have to supply some refined interaction rule for controlling the admissible transmission of booleans.

C-INP:	$\sum_{j \in J} y_j(x).P_j \xrightarrow{y_k z} P_k$	*if* $k \in J$

Table 3: Operational semantics for $\sum$-expressions

labeled transition semantics of $\mathbb{P}^{\mathsf{test}}$ is then determined as the smallest relation generated by the rules in Table 4 and Table 1, where we add the side-condition $z \in \mathbf{N}$ to the rules *INP* and *R-INP*; this additional side-condition prevents the standard input forms from receiving special names, and thus also from unintendedly using special names as channels.

After having presented the encodings into $\mathbb{T}$ in §4.2, we formalize in §4.4 the interpretation of those encodings as encodings into $\mathbb{P}$. The reader might object that these encodings can not be regarded as identical since $\mathbb{T}$ contains test-expressions. However, we can safely make this identification up to some notion of equivalence. As one might expect, this depends on the way that we actually use the additional primitives in our choice encodings: we only use the booleans on restricted channels, so they never become visible. The formal justification later on uses this fact explicitly (cf. §5.5).

4.2 Two choice encodings

This subsection contains two simple encodings of $\mathbb{S}$ into its choice-free fragment $\mathbb{T}$. Both encodings $\mathcal{C}[\![\ ]\!], \mathcal{D}[\![\ ]\!] : \mathbb{S} \to \mathbb{T}$ map terms of the source language $\mathbb{S}$ inductively into the target language $\mathbb{T}$. Since the encodings coincide on all constructors but choice, we use a common homomorphic scheme of definition, where $[\![\ ]\!]$ may denote either $\mathcal{C}[\![\ ]\!]$ or $\mathcal{D}[\![\ ]\!]$:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
[\![\,(x)P\,]\!] & \stackrel{\text{def}}{=} & (x)[\![\,P\,]\!] & [\![\,P_1|P_2\,]\!] & \stackrel{\text{def}}{=} & [\![\,P_1\,]\!] \mid [\![\,P_2\,]\!] \\
[\![\,\overline{y}z\,]\!] & \stackrel{\text{def}}{=} & \overline{y}z & [\![\,\mathbf{0}\,]\!] & \stackrel{\text{def}}{=} & \mathbf{0} \\
[\![\,y(x).P\,]\!] & \stackrel{\text{def}}{=} & y(x).[\![\,P\,]\!] & [\![\,!\,R\,]\!] & \stackrel{\text{def}}{=} & !\,[\![\,R\,]\!]
\end{array}$$

This scheme will also be reused later on for other encoding functions: if an encoding $\mathcal{X}[\![\ ]\!]$ *acts homomorphically on each constructor of* $\mathbb{P}$, then it is defined according to the scheme with $[\![\ ]\!]$ replaced by $\mathcal{X}[\![\ ]\!]$.

A source-level choice term is implemented by a particular target-level structure: for each choice expression, the translation

$$\left[\!\!\left[\, \sum_{j \in J} R_j \,\right]\!\!\right] \stackrel{\text{def}}{=} (l)\Big(\, \overline{l}\mathsf{t} \;\mid\; \prod_{j\in J} \mathit{Branch}_l\langle\, [\![\,R_j\,]\!]\,\rangle \Big)$$

installs a local *lock*—a message on l that carries a boolean name—that is only accessible for the parallel composition of its branches, which use the lock for running a mutual exclusion

TRUE:	$\mathsf{test}\ l\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \quad \xrightarrow{l\mathsf{t}} \quad P_1$
FALSE:	$\mathsf{test}\ l\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \quad \xrightarrow{l\mathsf{f}} \quad P_2$

Table 4: Operational semantics for test-expressions

protocol, as specified by the semantic rule *C-INP*. The protocol crucially depends on the invariant that at any time there is at most one boolean message available on l. If we manage to maintain this invariant, i.e., if it holds for all derivatives, then we are guaranteed that at any time the value of the lock is uniquely determined, which allows us to regard possible subsequent messages on l as representing the behavior of a 'determinate' lock that is accessible via l. The determinacy of the lock limits the nondeterminacy of the encoding solely to the concurrent race of the translations of the branches. Initially the lock carries t, so it represents the fact that the choice is not yet resolved; consequently, a committed choice will be recognizable by the lock carrying f.

Two slightly different ways of implementing the mutual exclusion protocol are given in the following subsections, differing only with respect to the possibility of undoing activities of branches, thus yielding two different definitions for the implementation Branch_l.

Divergence-free protocol

In the Introduction, we presented the following algorithm (note the use of test):

$$\mathit{Branch}_l\langle\, y_j(x).P_j\,\rangle \stackrel{\text{def}}{=} y_j(x)\,.\,\mathsf{test}\ l\ \mathsf{then}\ (\ \mathcal{C}[\![\, P_j\,]\!] \mid \bar{l}\mathsf{f}\ )\ \mathsf{else}\ (\ \overline{y_j}x \mid \bar{l}\mathsf{f}\ )$$

Every branch tests the lock *after* having received a value on its channel. If the lock carries t, then the testing branch proceeds with its continuation, otherwise it resends the message that it has consumed and terminates. Each time the lock is read, it is immediately reinstalled—independent of its former value—with the value f. Thus, at most one branch will ever be chosen, since only the first branch that reads the lock will read t.

In order to conveniently denote intermediate states in the branches of an encoding, we use the following abbreviations, where R on the left hand side matches the term $y(x).P$ such that y, x, P may be used on the right-hand side of the definitions.

$$\begin{aligned}
\mathit{Read}_l\langle\, R\,\rangle &\stackrel{\text{def}}{=} y(x).\mathit{Test}_l\langle\, R\,\rangle \\
\mathit{Test}_l\langle\, R\,\rangle &\stackrel{\text{def}}{=} \mathsf{test}\ \ l\ \ \mathsf{then}\ \ \mathit{Commit}_l\langle\, R\,\rangle\ \ \mathsf{else}\ \ \mathit{Abort}_l\langle\, R\,\rangle \\
\mathit{Commit}_l\langle\, R\,\rangle &\stackrel{\text{def}}{=} \bar{l}\mathsf{f} \mid P \\
\mathit{Abort}_l\langle\, R\,\rangle &\stackrel{\text{def}}{=} \bar{l}\mathsf{f} \mid \overline{y}x
\end{aligned}$$

Observe that $\mathit{Branch}_l\langle\, R\,\rangle = \mathit{Read}_l\langle\, R\,\rangle$ by definition, so a choice expression is translated by

$$\mathcal{C}\Big[\!\!\Big[\, \sum_{j\in J} R_j \,\Big]\!\!\Big] \quad \stackrel{\text{def}}{=} \quad (l)\Big(\ \bar{l}\mathsf{t} \ \mid\ \prod_{j\in J} \mathit{Read}_l\langle\, \mathcal{C}[\![\, R_j\,]\!]\,\rangle\ \Big) \qquad \text{where } l \text{ is fresh}$$

into the composition of its branches in *Read*-state and the lock carrying t.

Note that the $\mathcal{C}$-encoding does not add divergence to the behavior of source terms, as can be observed by inspection of the abbreviations: only finite τ-sequences are used in order to implement a choice, whereas τ-loops are not possible (we prove it formally in §5.8).

Protocol with undo-loops

We now define the other choice encoding. Its main difference from the encoding $\mathcal{C}[\![\ ]\!]$ is that a supposedly committed branch may change its mind and deny the commitment, releasing the lock and giving back the value it consumed from the environment. Let *internal choice* be:

$$P \oplus Q \stackrel{\text{def}}{=} (i)\ (\ \bar{i} \mid i.P \mid i.Q\ ) \qquad \text{where } i \text{ fresh}$$

The encoding $\mathcal{D}[\![\ ]\!]$ is then defined by modifying just the clause $\mathit{Test}_l\langle R \rangle$ of the $\mathcal{C}$-encoding by inserting an internal choice as follows:

$$\begin{aligned}\mathit{Test}_l\langle R \rangle &\stackrel{\text{def}}{=} \mathsf{test}\ l\ \mathsf{then}\ \mathit{Commit}_l\langle R \rangle \oplus \mathit{Undo}_l\langle R \rangle\ \mathsf{else}\ \mathit{Abort}_l\langle R \rangle \\ \mathit{Undo}_l\langle R \rangle &\stackrel{\text{def}}{=} \bar{l}\mathsf{t} \mid \overline{y}x\end{aligned}$$

Note that in contrast to the cases of *Commit*/*Abort*, where the lock's value is set to f, in the case of *Undo* it is crucial to set it to t, because in this case the choice is still not resolved. In order to have a fresh copy of the branch in initial state available after having undone an activity, we use replication of the *Read*-agents as the translation of branches: with $\mathit{Branch}_l\langle R \rangle = \mathbf{!}\,\mathit{Read}_l\langle R \rangle$ a choice expression is translated by:

$$\mathcal{D}\Big[\!\!\Big[\sum_{j\in J} R_j\Big]\!\!\Big] \stackrel{\text{def}}{=} (l)\Big(\ \bar{l}\mathsf{t} \mid \prod_{j\in J} \mathbf{!}\,\mathit{Read}_l\langle \mathcal{D}[\![R_j]\!] \rangle\ \Big) \qquad \text{where } l \text{ is fresh}$$

In their $\mathcal{D}$-translation, convergent branches of a choice term possibly engage in internal loops that may be used to restart a possibly committing branch from an initial state. This behavior might be considered problematic from an implementation point of view, but in the next subsection, we shall see that the $\mathcal{D}$-encoding is interesting despite its divergence.

4.3 An example

We highlight the difference between the two choice encodings by comparing a particular source term with both of its translations. Let

$$S \ =\ \overline{y_2}z \mid N \qquad \textit{where} \qquad N \ =\ R_1 + R_2 \qquad \textit{with} \qquad R_i \ =\ y_i(x).P_i$$

describe a binary choice in the presence of a single message matching the second of the branches, where P_1, P_2 are arbitrary terms in $\mathbb{P}$ (i.e., not containing choices or test-expressions). It turns out that both $S \not\approx \mathcal{C}[\![S]\!]$ and $S \approx \mathcal{D}[\![S]\!]$ hold, which imply that the $\mathcal{C}$-encoding neither preserves nor reflects weak bisimulation (Lemmas 4.3.3 and 4.3.4).

Divergence-free encoding

The transition systems of the above S and $\mathcal{C}[\![\, S \,]\!]$ can be depicted as follows. Since, for establishing bisimulations in the asynchronous π-calculus, input transitions are only considered in the context of arbitrary messages, we do not mention them explicitly in the pictures, but just the internal and output transitions. The dotted lines representing simulation relations are to be read from left to right; when the lines are vertical, simulation holds in both directions. We get

$$\begin{tikzcd}
\mathbb{S}: & N \arrow[d, dash, dotted, "\preccurlyeq\ \succcurlyeq"] & S \arrow[l, "\overline{y_2}z"'] \arrow[rr, "\tau"] \arrow[d, dash, dotted, "\preccurlyeq\ \succcurlyeq"] \arrow[dr, dash, dotted, "\succcurlyeq"] & & P_2\{^z\!/_x\} \arrow[d, dash, dotted, "\preccurlyeq\ \succcurlyeq"] \arrow[dl, dash, dotted, "\succcurlyeq"'] \\
\mathbb{T}: & \mathcal{C}[\![\, N \,]\!] & \mathcal{C}[\![\, S \,]\!] \arrow[l, "\overline{y_2}z"'] \arrow[r, "\tau"] & C \arrow[r, "\tau"] & C_2
\end{tikzcd}$$

where (letting $B_i = \mathit{Read}_l\langle \mathcal{C}[\![\, R_i \,]\!] \rangle$):

$$\begin{array}{rcll}
\mathcal{C}[\![\, S \,]\!] & = & \overline{y_2}z \mid \mathcal{C}[\![\, N \,]\!] & \\
\mathcal{C}[\![\, N \,]\!] & = & (l)\Big(\ \bar{l}\mathsf{t} \mid B_1 \mid & B_2\ \Big) \\
C & = & (l)\Big(\ \bar{l}\mathsf{t} \mid B_1 \mid & \mathit{Test}_l\langle \mathcal{C}[\![\, R_2 \,]\!] \rangle\ \{^z\!/_x\}\ \Big) \\
C_2 & = & (l)\Big(\ \mathbf{0} \mid B_1 \mid & \mathit{Commit}_l\langle \mathcal{C}[\![\, R_2 \,]\!] \rangle\ \{^z\!/_x\}\ \Big) \\
& \equiv & \underbrace{(l)\Big(\ \bar{l}\mathsf{f} \mid B_1\ \Big)}_{\approx \mathbf{0} \text{ (see Lemma 5.2.2)}} & \mid \mathcal{C}[\![\, P_2 \,]\!]\{^z\!/_x\}
\end{array}$$

We may phrase the intermediate state C as having *partially committed*: at that stage of commitment, it is already clear that *one* of the branches will eventually be chosen, but it is not yet clear if it will be the activated branch (as we chose in the computation path with C_2) or if it will be the competing branch (still waiting at y_1 for some value), since there might still be a suitable message provided by the context which could activate the latter and afterwards pre-empt the former. As a consequence, the state C does not directly correspond to any of the source terms with respect to weak bisimulation, which implies:

Fact 4.3.1 *$S \not\approx \mathcal{C}[\![\, S \,]\!]$.*

On the other hand, the observation that $S \succcurlyeq C \succcurlyeq P_2\{^z\!/_x\}$ with $S \Rightarrow P_2\{^z\!/_x\}$ in the above example suggests *coupled simulation* as an appropriate notion of correctness for the $\mathcal{C}$-encoding.

Divergent encoding

The corresponding transition systems of S and $\mathcal{D}[\![\,S\,]\!]$, again omitting input transitions, are

$$\begin{tikzcd}
\mathbb{S}: & N \arrow[d, dash, dotted, "\approx"] & S \arrow[l, "\overline{y_2}z"'] \arrow[rrr, "\tau"] \arrow[d, dash, dotted, "\approx"] \arrow[dr, dash, dotted, "\approx"] \arrow[drr, dash, dotted, "\approx"] & & & P_2\{^z/_x\} \arrow[d, dash, dotted, "\approx"] \\
\mathbb{T}: & \mathcal{D}[\![\,N\,]\!] & \mathcal{D}[\![\,S\,]\!] \arrow[l, "\overline{y_2}z"'] \arrow[r, "\tau"] & D \arrow[r, "\tau"] & D' \arrow[r, "\tau"] \arrow[ll, bend left, "\tau"] & D_2
\end{tikzcd}$$

where (letting $B_i = \mathbf{!}\, \mathit{Read}_l\langle \mathcal{D}[\![\,R_i\,]\!]\rangle$):

$$\begin{aligned}
\mathcal{D}[\![\,S\,]\!] &= \overline{y_2}z \mid \mathcal{D}[\![\,N\,]\!] \\
\mathcal{D}[\![\,N\,]\!] &= (l)\Big(\, \bar{l}\mathsf{t} \mid B_1 \mid B_2 \Big) \\
D \quad &= (l)\Big(\, \bar{l}\mathsf{t} \mid B_1 \mid B_2 \mid \quad \mathit{Test}_l\langle \mathcal{D}[\![\,R_2\,]\!]\rangle \;\{^z/_x\} \Big) \\
D' \quad &= (l)\Big(\, \mathbf{0} \mid B_1 \mid B_2 \mid \quad \big(\mathit{Commit}_l\langle \mathcal{D}[\![\,R_2\,]\!]\rangle \oplus \mathit{Undo}_l\langle \mathcal{D}[\![\,R_2\,]\!]\rangle\big)\{^z/_x\} \Big) \\
&= (l)\Big(\, \mathbf{0} \mid B_1 \mid B_2 \mid \quad \big(\mathcal{D}[\![\,P_2\,]\!]\{^z/_x\} \mid \bar{l}\mathsf{f}\big) \oplus \big(\overline{y_2}z \mid \bar{l}\mathsf{t}\big) \Big) \\
D_2 \quad &\equiv \underbrace{(l)\Big(\bar{l}\mathsf{f} \mid B_1 \mid B_2\Big)}_{\approx \mathbf{0} \text{ (see Lemma 5.7.1)}} \mid \mathcal{D}[\![\,P_2\,]\!]\{^z/_x\}
\end{aligned}$$

The internal undo-transition from D' back to $\mathcal{D}[\![\,S\,]\!]$ is essential in proving that the intermediate states D and D' are actually weakly bisimilar to S, since only by internally looping back to the initial state can they simulate all of S's behavior; that possibility is lacking in the $\mathcal{C}$-encoding.

Proposition 4.3.2 $S \approx \mathcal{D}[\![\,S\,]\!]$.

Note another aspect of the above computation paths of $\mathcal{C}[\![\,S\,]\!]$ and $\mathcal{D}[\![\,S\,]\!]$: Whereas the step from $\mathcal{C}[\![\,S\,]\!]$ to C uses up the component B_2, its correspondent survives in the step from $\mathcal{D}[\![\,S\,]\!]$ to D since, there, it is a replication.

Full abstraction?

We can now use S to prove that the $\mathcal{C}$-encoding is not fully abstract with respect to weak bisimulation. Note that both $\mathcal{C}[\![\;]\!]$ and $\mathcal{D}[\![\;]\!]$, extended in the obvious way for `test`-clauses, act as an identity on terms in $\mathbb{T}$, which implies that $\mathcal{C}[\![\,\mathcal{C}[\![\,S\,]\!]\,]\!] = \mathcal{C}[\![\,S\,]\!]$ and $\mathcal{C}[\![\,\mathcal{D}[\![\,S\,]\!]\,]\!] = \mathcal{D}[\![\,S\,]\!]$.

Weak bisimulation is not reflected by the $\mathcal{C}$-encoding.

Lemma 4.3.3 *$\mathcal{C}[\![\, S_1 \,]\!] \approx \mathcal{C}[\![\, S_2 \,]\!]$ does not imply $S_1 \approx S_2$.*

Proof: Let $S_1 = S$ be the example above and $S_2 = \mathcal{C}[\![\, S \,]\!]$ its translation. By definition, we have $\mathcal{C}[\![\, S_2 \,]\!] = \mathcal{C}[\![\, \mathcal{C}[\![\, S \,]\!] \,]\!] = \mathcal{C}[\![\, S \,]\!] = \mathcal{C}[\![\, S_1 \,]\!]$, but $S_2 \not\approx S$. □

Weak bisimulation is also not preserved by the $\mathcal{C}$-encoding.

Lemma 4.3.4 *$S_1 \approx S_2$ does not imply $\mathcal{C}[\![\, S_1 \,]\!] \approx \mathcal{C}[\![\, S_2 \,]\!]$.*

Proof: Let $S_1 = S$ and $S_2 = \mathcal{D}[\![\, S \,]\!]$, so that $S_1 \approx S_2$. Suppose, for a contradiction, that $\mathcal{C}[\![\ ]\!]$ preserved $\approx$; then, by definition, we would have $\mathcal{C}[\![\, S \,]\!] = \mathcal{C}[\![\, S_1 \,]\!] \approx \mathcal{C}[\![\, S_2 \,]\!] = \mathcal{C}[\![\, \mathcal{D}[\![\, S \,]\!] \,]\!] = \mathcal{D}[\![\, S \,]\!] \approx S$, which contradicts Fact 4.3.1. □

4.4 Expanding the encodings

In §4.1, we introduced the intermediate language $\mathbb{P}^{\mathsf{test}}$ as a for the convenient presentation of choice encodings. We justify the use of $\mathbb{P}^{\mathsf{test}}$ by translating it itself into the language $\mathbb{P}$.

We first sketch the idea of expanding boolean messages and test-expressions into $\mathbb{P}$ by using 2-adic messages. It is directly inspired from a standard protocol for encoding the behavior of persistent boolean values [Mil93] adapted to the needs of the example, where the boolean messages correspond to ephemeral boolean variables. The encoding function $\widetilde{\mathcal{B}}[\![\ ]\!]$ acts homomorphically on every constructor of $\mathbb{P}$ that does not involve booleans; for those constructors that apply to or just mention booleans, $\widetilde{\mathcal{B}}[\![\ ]\!]$ is defined by

$$\begin{array}{rcl}
\widetilde{\mathcal{B}}[\![\, \bar{l}\mathsf{t} \,]\!] & \stackrel{\text{def}}{=} & l(t,f).\bar{t} \\
\widetilde{\mathcal{B}}[\![\, \bar{l}\mathsf{f} \,]\!] & \stackrel{\text{def}}{=} & l(t,f).\overline{f} \\
\widetilde{\mathcal{B}}[\![\, \mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q \,]\!] & \stackrel{\text{def}}{=} & (t)(f)(\ \bar{l}\langle t,f\rangle \mid t.\widetilde{\mathcal{B}}[\![\, P \,]\!] \mid f.\widetilde{\mathcal{B}}[\![\, Q \,]\!]\ )
\end{array}$$

where $\bar{l}\langle t,f\rangle$ and $l(t,f).P$ denote 2-adic communication (cf. [Mil93]). In contrast to the higher-level test-notation, the $\widetilde{\mathcal{B}}$-encoding causes one additional τ-step for each test.

We still have not yet arrived at the language $\mathbb{P}$ due to the use of 2-adic messages in the $\widetilde{\mathcal{B}}$-encoding; note that we do not need the expressive power of the full polyadic π-calculus, here. To get rid of them, we may use a technique for asynchronously encoding 2-adic messages [HT91] that was called 'zip-lock' technique by Odersky [Ode95a]. The encoding function $\mathcal{Z}[\![\ ]\!]$ acts homomorphically on every constructor of $\mathbb{P}$; for 2-adic messages and inputs,

$$\begin{array}{rcl}
\mathcal{Z}[\![\, \bar{l}\langle t,f\rangle \,]\!] & \stackrel{\text{def}}{=} & (u)(\bar{l}u \mid\ u(v).(\ \overline{v}t \mid\ u(w).\overline{w}f)) \\
\mathcal{Z}[\![\, l(t,f).P \,]\!] & \stackrel{\text{def}}{=} & l(u).(v)(\overline{u}v \mid v(t).(w)(\overline{u}w \mid w(f).\mathcal{Z}[\![\, P \,]\!]))
\end{array}$$

yields monadic processes, where the names u, v, w are assumed not to occur free in P. We could have used fewer freshly created names by reusing u for different purposes, but the number of necessary internal actions cannot be reduced, since one (on l) is needed to establish the connection, two (on v, w) for transferring the data, and another two (on u) for transmitting the local channels and also serving as internal acknowledgments in order to maintain the order of the data by transmitting them in a 'zip-lock' way.

By using the zip-lock encoding, we give an expanded $\mathcal{B}$-encoding into the monadic target language $\mathbb{P}$, where we also use 'void' names _ that are merely used as signal objects and could be instantiated with any name since they are never used as channels.

$$\begin{aligned}
\mathcal{B}\llbracket\, \bar{l}\mathsf{t} \,\rrbracket &\stackrel{\text{def}}{=} \mathcal{Z}\llbracket\, l(t,f).\bar{t} \,\rrbracket \\
\mathcal{B}\llbracket\, \bar{l}\mathsf{f} \,\rrbracket &\stackrel{\text{def}}{=} \mathcal{Z}\llbracket\, l(t,f).\bar{f} \,\rrbracket \\
\mathcal{B}\llbracket\, \mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q \,\rrbracket &\stackrel{\text{def}}{=} (t)(f)(\ \mathcal{Z}\llbracket\, \bar{l}\langle t,f\rangle \,\rrbracket \mid t.\mathcal{B}\llbracket\, P \,\rrbracket \mid f.\mathcal{B}\llbracket\, Q \,\rrbracket\ )
\end{aligned}$$

Obviously, a language with boolean messages and test-expressions as primitives will be much clearer than the $\mathcal{B}$-translations, when used in order to write down an encoding function.

Finally, we are ready to give the full encodings $\mathcal{C}_{\lesssim}\llbracket\ \rrbracket$ and $\mathcal{D}_{\lesssim}\llbracket\ \rrbracket$ by composing the earlier encodings into the intermediate target language $\mathbb{T}$ with the expanding encoding $\mathcal{B}\llbracket\ \rrbracket$:

$$\begin{aligned}
\mathcal{C}_{\lesssim}\llbracket\ \rrbracket &\stackrel{\text{def}}{=} (\mathcal{B}\circ\mathcal{C})\llbracket\ \rrbracket \\
\mathcal{D}_{\lesssim}\llbracket\ \rrbracket &\stackrel{\text{def}}{=} (\mathcal{B}\circ\mathcal{D})\llbracket\ \rrbracket
\end{aligned}$$

which are indeed encodings of $\mathbb{P}^{\Sigma}$ into its choice-free fragment $\mathbb{P}$.

$$\begin{tikzcd}
\mathbb{S} = & \mathbb{P}^{\Sigma} \arrow[d, "\mathcal{C}_{\lesssim}\llbracket\ \rrbracket{,}\mathcal{D}_{\lesssim}\llbracket\ \rrbracket"'] \arrow[dr, dotted, "\mathcal{C}\llbracket\ \rrbracket{,}\mathcal{D}\llbracket\ \rrbracket"] & & \\
& \mathbb{P} & \mathbb{P}^{\mathsf{test}} \arrow[l, dotted, "\mathcal{B}\llbracket\ \rrbracket"] & = \mathbb{T}
\end{tikzcd}$$

We can 'safely' use primitive test-forms within the encoding functions since, as we will show in §5.5, the following property holds:

$$\text{for all } S \in \mathbb{S}\ :\ \text{for all } \mathcal{C}\llbracket\, S \,\rrbracket \ \longrightarrow^{*} T\ :\ T \lesssim \mathcal{B}\llbracket\, T \,\rrbracket$$

i.e., the encoding $\mathcal{B}\llbracket\ \rrbracket$ 'expands' certain $\mathbb{T}$-terms in the sense of Definition 2.3.5: T and $\mathcal{B}\llbracket\, T \,\rrbracket$ are weakly bisimilar, but the latter uses more τ-steps. The proof of this property relies on the fact that any use of booleans within the encoding $\mathcal{C}\llbracket\ \rrbracket$ is restricted for an outside observer of a choice expression. The same property holds of the encoding $\mathcal{D}\llbracket\ \rrbracket$.

4.5 Discussion

As the distinguishing example in §4.3 indicates, for reasoning about the choice encoding's behavioral correctness, we are able to compare source terms S and their translations $[\![S]\!]$ directly; the visible labels of source and target transitions are the same. Furthermore, by the arguments of the previous section, both encodings can be regarded as endomorphic mappings where the target $\mathbb{T}$ is a fragment of the source $\mathbb{S}$, at least up to expansion of translations.

With respect to operational correspondence, the choice encodings represent the class of non-prompt translations where, as in the motivation of the soundness property $\mathfrak{S}$, committing steps of translations are preceded by administrative steps. Those pre-administrative steps can not be simply defined away (by using administrative normal forms) since, in general, they are not confluent: imagine a process containing two choices that compete for a single message; both choices could evolve by consuming the message, but each would pre-empt the other.

In the first author's PhD thesis [Nes96], the discussion of appropriate notions of correctness as well as a discussion of possible (correct) and impossible (incorrect) variants of the proposed choice encodings are carried out in considerably more detail.

5 Correctness proof by decoding

In this section, we prove the correctness of the $\mathcal{C}$-encoding with respect to coupled simulation equivalence (§5.2–§5.6) and sketch the corresponding proof for the $\mathcal{D}$-encoding with respect to weak bisimulation equivalence (§5.7). The main contribution of this section is a proof notation and technique, with decoding functions, which we use to exhibit that source terms and their translations are in fact equivalent. We also include a proof that the $\mathcal{C}_{\lesssim}$-encoding is divergence-free (§5.8).

5.1 Overview

Since the choice encodings are not prompt, we have to explicitly deal with the behavior of intermediate states and relate them to source terms with equivalent behavior. Moreover, in the case of the $\mathcal{C}_{\lesssim}$-encoding, an intermediate state may be partially committed, so we may have to relate it to two different source terms. By definition, partial commitments are absent in source terms; thus, a partially committed derivative of a translation can only be related to source terms which represent either its *reset* or its *completion*.

Technically, we are going to build the coupled simulation constructively as a pair of *decoding* functions from target terms to source terms. However, intermediate states are target terms that have lost the structure of being literal translations of source terms; thus, it is impossible to denote the source term from which an intermediate target term derives without some knowledge of its derivation history.

We therefore introduce *annotated* source terms (§5.2) as abbreviations for derivatives of their translations. An annotated term shows its source-level choice structure while providing information about which target state its choices inhabit, using a representation of its derivation history that is constructed from an operational semantics of annotated choice. We formally introduce the language $\mathbb{A}$ of annotated source terms and use it as follows for the investigation of the correctness of the $\mathcal{C}_{\lesssim}$-encoding:

Factorization (§5.3) Annotated source terms represent abbreviations of target terms. We define an *annotation* encoding $\mathcal{A}$ mapping source terms to abbreviations and a *flattening* encoding $\mathcal{F}$ expanding abbreviations to target terms.

$$\begin{tikzcd}
\mathbb{S} \arrow[r, "\mathcal{A}[\![\ ]\!]"] \arrow[d, dotted] \arrow[dr, "\mathcal{C}[\![\ ]\!]"'] & \mathbb{A} \arrow[d, "\mathcal{F}[\![\ ]\!]"] \\
\mathbb{P} & \mathbb{T} \arrow[l, dotted]
\end{tikzcd}$$

Decoding (§5.4) Annotated source terms deal with partial commitments explicitly. We define two *de*coding functions $\mathcal{U}$ of annotated terms back into source terms, where $\mathcal{U}_\flat$ resets and $\mathcal{U}_\sharp$ completes partial commitments.

Expansion (§5.5) Since the target language $\mathbb{T}$ is a language extended with booleans, we show that their use in our setting is rather well-behaved according to the *expanding* encoding $\mathcal{B}$.

$$\begin{tikzcd}
\mathbb{S} \arrow[d, dotted] & \mathbb{A} \arrow[l, shift right, "\mathcal{U}_\sharp[\![\ ]\!]"'] \arrow[l, shift left, "\mathcal{U}_\flat[\![\ ]\!]"] \arrow[d, dotted] \\
\mathbb{P} & \mathbb{T} \arrow[l, "\mathcal{B}[\![\ ]\!]"]
\end{tikzcd}$$

The factorization, the decodings, and the expansion enjoy several nice properties:

1. $\mathcal{F}$ is a strong bisimulation between abbreviations and boolean target terms.
2. $(\mathcal{U}_\flat,\mathcal{U}_\sharp)$ is a coupled simulation between abbreviations and source terms.
3. $\mathcal{B}$, i.e., a variant of it, is an expansion for booleans in target terms.

Those can be combined to provide a coupled simulation on $\mathbb{S} \times \mathbb{P}$. The observation that every source term S and its translation $\mathcal{C}_{\lesssim}[\![\, S \,]\!]$ are related by this relation concludes the proof of coupled-simulation-correctness of the $\mathcal{C}_{\lesssim}$-encoding (§5.6).

Simplifications due to homomorphic encodings and decodings Many of the proofs in this section have in common that they exhibit particular transitions of terms by constructing appropriate inference trees either

- from the inductive structure of (annotated) terms, or
- by simply replacing some leafs in the inference trees of their encodings or decodings.

Since the $\mathcal{A}, \mathcal{F}, \mathcal{U}_\flat, \mathcal{U}_\sharp, \mathcal{B}$-functions are each defined homomorphically on every constructor of $\mathbb{P}$ according to the scheme in §4, there is a strong syntactic correspondence between terms and their respective translations, and, as a consequence, there is also a strong correspondence between transition inference trees. More precisely, since in transitions involving choice

- there is at most one application of a choice rule, and
- an application of the choice rule always represents a leaf in the inference tree,

it suffices for all proofs to regard choice terms in isolation.

When looking for simulations between terms and their translations (as in the proofs of Proposition 5.3.4, Lemmas 5.3.1, 5.4.2 and 5.4.3), a transition which does not involve choice rules is trivially simulated via the identical inference tree by the translated term. A transition which emanates from a choice term inside a term context, and which may be simulated by the translated choice term in isolation, will also be derivable by the same inference tree, except for the leaf being adapted to the simulating transition. Note that possible side-conditions are not critical, if the simulation has been successful for the choice term itself.

When (as in the proofs 5.4.7 and 5.4.6) looking for internal transitions of a particular term, generated by occurrences of choice, it suffices to use choices in isolation since τ-steps are passed through arbitrary contexts without condition.

5.2 Annotated choice

This subsection introduces an annotated variant of choice, which provides abbreviations for all derivatives of $\mathcal{C}$-translations. Its shape reveals the high-level structure, but it exhibits low-level operational behavior. The attached annotations record essential information about the low-level derivation history.

According to the definition in §4, the computations of individual branches of a choice will pass through basically three different states: *Read*, *Test*, and one of *Commit* or *Abort* as the final state. The state of each branch is completely determined by the result of two inputs:

- the value (if any) which is currently carried by the channel, and
- the boolean (if any) which has been assigned by acquiring the lock.

For the representation of that information for a J-indexed choice, we use

- a partial function $\mathrm{v} : J \rightharpoonup \mathbf{V}$, mapping choice indices to values, and
- a possibly empty set $B \subseteq J$ of choice indices such that $B \cap \mathsf{dom}(\mathrm{v}) = \emptyset$.

In the context of a particular J-indexed choice with branches $R_j = y_j(x).P_j$ for each $j \in J$, the definedness of v_j (the value $\mathrm{v}(j)$ held by branch j) means that a value has been read from the environment, but has not yet either led to a commitment of branch j or been reinjected into the environment. The set B records those branches which have already accessed the

READ:	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B}$	$\xrightarrow{y_k z}$	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}+(k\mapsto z)}_{B}$	*if* $k \in J\backslash(V\cup B)$
COMMIT:	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\emptyset}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{\{k\}} \mid P_k\{^{\mathrm{v}_k}\!/_x\}$	*if* $k \in V$
ABORT:	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B\neq\emptyset}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{B+k} \mid \overline{y_k}\,\mathrm{v}_k$	*if* $k \in V$

Table 5: Early transition semantics for annotated choice

boolean lock. An empty set B means that *none* of the branches has yet been chosen, i.e., that the choice has not (yet) committed. Then, with the abbreviation $V := \mathsf{dom}(\mathrm{v})$, the state of each individual branch can be retrieved from the annotations v and B. Branch $k \in J$ is in

- *Read*-state if $k \in J\backslash(V\cup B)$
- *Test*-state if $k \in V$
- *Commit/Abort*-state if $k \in B$.

The state of the whole choice is completely determined by the states of its branches.

Definition 5.2.1 (Annotated choice) *Let J be a set of indices. Let $R_j = y_j(x).P_j$ be input prefixes for $j \in J$. Let $\mathrm{v} : J \rightharpoonup \mathbf{V}$ and $B \subseteq J$ with $B \cap V = \emptyset$, where $V := \mathsf{dom}(\mathrm{v})$. Then,*

$$\sum_{j\in J} R_j \qquad \textit{and} \qquad \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B}$$

are referred to as bare *and* annotated *choice, respectively.*

Annotated choice is given the operational semantics in Table 5. The dynamics of annotated choice mimic precisely the behavior of the intended low-level process. *READ* allows a branch k in *Read*-state ($k \in J\backslash(V\cup B)$) to optimistically consume a message.[2] If the choice is not yet resolved ($B = \emptyset$), *COMMIT* specifies that an arbitrary branch k in *Test*-state ($k \in V$) can immediately evolve into its *Commit*-state. i.e., trigger its continuation process P_k. After the choice is resolved ($B \neq \emptyset$), *ABORT* allows any branch k in *Commit*-state ($k \in V$) to evolve into its *Abort*-state release their consumed messages. Intuitively, by reading the

[2]Compared to its late counterpart, the early instantiation scheme may be more intuitive for showing that a particular value has entered the choice system. Furthermore, we do not have to deal with α-conversion of homonym bound names in different branches. Nevertheless, the further development in this paper does not depend on this decision. Note also that early and late bisimulation coincide in our setting (cf. §2.3).

lock, a branch immediately leaves the choice system and exits. Therefore, annotated choice only contains branches in either *Read*- or *Test*-state.

We distinguish three cases for choice constructors that are important enough to give them names: *initial* for $V = \emptyset = B$, *partial* for $V \neq \emptyset$ and $B = \emptyset$, and *committed* for $B \neq \emptyset$. Note that both initial and partial choice contain all branches, whereas committed choice never does; it will even become empty, once all branches have reached their final state ($B = J$).

Committed choice exhibits a particularly interesting property: its branches in *Test*-state already have consumed a message which they will return after recognizing, by internally testing the lock, that the choice is already committed; its branches in *Read*-state are still waiting for values to be consumed and—since the choice is resolved and the lock carries f—immediately resent after an internal step. Processes with such receive-and-resend behavior are weakly bisimilar to $\mathbf{0}$ and were called *identity receptors* by Honda and Tokoro [HT92]. In fact, a stronger property holds:

Lemma 5.2.2 $\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B\neq\emptyset} \gtrsim \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j$.

Proof: Let M be an arbitrary m-ary composition of messages. We show that

$$\mathcal{R} \stackrel{\text{def}}{=} \Big\{ \Big(\underbrace{M \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B\neq\emptyset}}_{\text{LHS}} \,,\; \underbrace{M \mid \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j}_{\text{RHS}} \Big) \;\Big|\; \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B} \in \mathbb{A} \Big\}$$

is an expansion.

case *internal steps* According to the operational semantics, τ-transitions are only possible for LHS. Since RHS does not exhibit τ-transitions, we show that for all LHS $\xrightarrow{\tau}$ LHS$'$, we have (LHS$'$, RHS) $\in \mathcal{R}$. There are two subcases: either (1) via an *ABORT*-step of choice, or (2) via choice consuming an M-message due to *READ*.

case *ABORT* Since $B \neq \emptyset$, for $k \in V$, via *ABORT*, followed by m times PAR_1, we have

$$\text{LHS} = M \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B} \xrightarrow{\tau}\equiv M \mid \overline{y_k}\,\mathrm{v}_k \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{B+k} =: \text{LHS}'$$

Since k's values are erased from v, we observe that the $\mathcal{R}$-correspondent of LHS$'$—note that it is an admissible left hand side for $\mathcal{R}$—as defined by

$$M \mid \overline{y_k}\,\mathrm{v}_k \mid \prod_{k\neq j\in V} \overline{y_j}\,\mathrm{v}_j = M \mid \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j$$

coincides with RHS.

case *READ* For $M \equiv N \mid \overline{y}z$ and $k \in J\backslash(V\cup B)$ such that $y = y_k$, let $\mathrm{v}' = \mathrm{v} + (k\mapsto z)$. Then, via rule *READ* for the choice part, *OUT* for the indicated y-message, and rule *COM* to derive the τ-transition from the former visible transitions, we have

$$\mathrm{LHS} = N \mid \overline{y}z \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_B \xrightarrow{\tau} N \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}'}_B =: \mathrm{LHS}'$$

By definition, the $\mathcal{R}$-correspondent of LHS′, is of the form

$$N \mid \prod_{j\in V'} \overline{y_j}\,\mathrm{v}_j = N \mid \overline{y_k}\,\mathrm{v}_k \mid \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j = M \mid \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j$$

which coincides with RHS.

case *output steps* According to the operational semantics, output-transitions are possible for each message in M, i.e., for both LHS and RHS. The (strong) bisimulation behavior for that case is trivial. LHS does not exhibit further immediate outputs. In contrast, RHS allows outputs for each message in $\prod_{j\in V} \overline{y_j}\,\mathrm{v}_j$, i.e., for $k \in V$, via *OUT* and *PAR*, we have

$$\mathrm{RHS} = M \mid \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j \xrightarrow{\overline{y_k}\,\mathrm{v}_k} M \mid \prod_{k\neq j\in V} \overline{y_j}\,\mathrm{v}_j =: \mathrm{RHS}'$$

Due to *ABORT*, we derive an internal step and afterwards release the message $\overline{y_k}\,\mathrm{v}_k$

$$\begin{aligned} \mathrm{LHS} = M \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_B &\xrightarrow{\tau} M \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{B+k} \mid \overline{y_k}\,\mathrm{v}_k \\ &\xrightarrow{\overline{y_k}\,\mathrm{v}_k} M \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{B+k} =: \mathrm{LHS}' \end{aligned}$$

which weakly simulates the transition of RHS. We observe that $(\mathrm{LHS}', \mathrm{RHS}') \in \mathcal{R}$. □

Corollary 5.2.3 (Inertness) $\big(\sum_{j\in J} R_j\big)^{\emptyset}_{B\neq\emptyset} \gtrsim \mathbf{0}$.

From an asynchronous observer's point of view, committed choice behaves exactly like the composition of the messages that are held by branches in *Test*-state, except that it involves additional internal computation. Note that a standard (synchronous) observer, which may detect inputs, would be able to tell the difference.

Since the purpose of annotated choice is to keep track of which low-level actions belong to the same high-level choice, we introduce a language of *annotated processes*. The set $\mathbb{P}^{(\Sigma)}$ such processes is generated by adding a clause for $(\sum)$-expressions to the grammar of $\mathbb{P}$:

$$P \;::=\; \ldots \quad\mid\quad \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_B$$

The operational semantics of $\mathbb{P}^{(\Sigma)}$ is given by the rules in Table 1 and Table 5.

5.3 Factorization

We now introduce the components of a factorization for the encoding $\mathcal{C}[\![\ ]\!]$: an *annotation* encoding $\mathcal{A}[\![\ ]\!]$, and a *flattening* encoding $\mathcal{F}[\![\ ]\!]$.

$$\begin{tikzcd} \mathbb{S} \arrow[r, "\mathcal{A}[\![\ ]\!]"] \arrow[dr, dotted, "\mathcal{C}[\![\ ]\!]"'] & \mathbb{A} \arrow[d, "\mathcal{F}[\![\ ]\!]"] & :\subset \mathbb{P}^{(\Sigma)} \\ & \mathbb{T} & \end{tikzcd}$$

An intermediate sublanguage $\mathbb{A}$ will be characterized as the sublanguage of $\mathbb{P}^{(\Sigma)}$ that precisely contains the derivatives of $\mathcal{A}$-translations. We write $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{F}$ for the encoding functions considered as relations.

Annotation.

The encoding $\mathcal{A}[\![\ ]\!] : \mathbb{S} \to \mathbb{P}^{(\Sigma)}$ acts homomorphically on every constructor except for choice according to the scheme in §4. The latter case is given by

$$\mathcal{A}\left[\!\!\left[\sum_{j\in J} R_j \right]\!\!\right] \stackrel{\text{def}}{=} \Big(\sum_{j\in J} \mathcal{A}[\![R_j]\!]\Big)^{\emptyset}_{\emptyset}$$

which translates choices into their annotated counterparts with all branches in initial state. The following lemma represents a first simple operational completeness statement for $\mathcal{A}[\![\ ]\!]$.

Lemma 5.3.1 (Completeness) *$\mathcal{A}$ is a weak simulation up to expansion.*

Proof: We show the proof for weak *synchronous* simulation, which implies the asynchronous case. The proof is by structural induction on $S \in \mathbb{S}$ and transition induction on $S \;\to S'$. By the simplification discussed in Section 5.1, it suffices to regard the case

$$S = \sum_{j\in J} R_j$$

where, according to the rules in Table 3, there is only one subcase.

case *C-INP* For $k \in J$, with $\displaystyle\sum_{j\in J} R_j \xrightarrow{y_k z} P_k\{^z\!/_x\}$,

there is always a weakly simulating sequence by *READ* and *COMMIT*

$$
\begin{array}{rcl}
\mathcal{A}\left[\!\!\left[\sum_{j\in J} R_j \right]\!\!\right] = \big(\sum_{j\in J} \mathcal{A}[\![\, R_j \,]\!]\big)^{\emptyset}_{\emptyset} & \xrightarrow{y_k z} & \big(\sum_{j\in J} \mathcal{A}[\![\, R_j \,]\!]\big)^{(k\mapsto z)}_{\emptyset} \\
& \xrightarrow{\tau} & \mathcal{A}[\![\, P_k \,]\!]\{^z\!/_x\} \mid \big(\sum_{j\in J} \mathcal{A}[\![\, R_j \,]\!]\big)^{\emptyset}_{\{k\}} \\
& \gtrsim & \mathcal{A}[\![\, P_k \,]\!]\{^z\!/_x\} \mid \mathbf{0} \\
& \equiv & \mathcal{A}[\![\, P_k\{^z\!/_x\} \,]\!]
\end{array}
$$

where the expansion relation holds due to Lemma 5.2.2. □

Intermediate language.

Terms in the target of $\mathcal{A}[\![\ ]\!]$ and also their derivatives are of a particular restricted form, which can be made precise by characterizing the possible shape of occurrences of choice terms. Let a subterm be called *guarded* when it occurs as a subterm of a guard. The basic syntactic properties of the intermediate language then are:

- All occurrences of choice are annotated.
- All guarded occurrences of choice are initial.
- Unguarded occurrences of choice may be initial, partial, or committed.

Later on in this paper (for the proofs of the Lemmas 5.4.6 and 5.4.8), we need these properties in order to conclude that no guarded annotated choice is in an intermediate state.

Therefore, let $\mathbb{A}$ denote the sublanguage of terms in $\mathbb{P}^{(\Sigma)}$ that satisfy the above syntactic requirements; we give an inductive grammar for generating appropriate terms by two levels. One level generates terms I with only initially annotated occurrences of choice terms

$$
\begin{array}{rcl}
G & ::= & y(x).I \\
I & ::= & \mathbf{0} \quad \big| \quad \overline{y}z \quad \big| \quad G \quad \big| \quad (y)I \quad \big| \quad I|I \quad \big| \quad \mathbf{!}\,G \quad \big| \quad \big(\sum_{j\in J} G_j\big)^{\emptyset}_{\emptyset}
\end{array}
$$

Another level on top generates terms A which allow active (top-level) choices to be in intermediate state, but guarded occurrences of choice only to be initial, as specified by I:

$$
\begin{array}{rcl}
R & ::= & y(x).I \\
A & ::= & \mathbf{0} \quad \big| \quad \overline{y}z \quad \big| \quad R \quad \big| \quad (y)A \quad \big| \quad A|A \quad \big| \quad \mathbf{!}\,R \quad \big| \quad \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B}
\end{array}
$$

where $\mathrm{v} : J \rightharpoonup \mathbf{V}$ and $B \subseteq J$ for arbitrary index set J, as usual. A term $A \in \mathbb{A}$ is called *partially committed* (or *partial*), if it contains at least one occurrence of partial choice, and *fully committed* (or *full*), otherwise.

An important property of $\mathbb{A}$ is that it is transition-closed.

Lemma 5.3.2 *For all $A \in \mathbb{A}$, if $A \xrightarrow{\mu} A'$, then $A' \in \mathbb{A}$.*

Proof: By inspection of the rules in Table 5. $\square$

Flattening.

The encoding $\mathcal{F}[\![\ ]\!] : \mathbb{P}^{(\Sigma)} \to \mathbb{T}$ acts homomorphically on every constructor but annotated choice according to the scheme in §4. For annotated choice, the translation

$$\mathcal{F}\Big[\!\!\Big[\big(\sum_{j \in J} R_j \big)_B^{\mathrm{v}} \Big]\!\!\Big] \stackrel{\text{def}}{=} (l)\Big(\ \bar{l}\mathsf{b} \ \ | \prod_{j \in J \setminus (V \cup B)} Read_l\langle \mathcal{F}[\![R_j]\!] \rangle \ \ | \prod_{j \in V} Test_l\langle \mathcal{F}[\![R_j]\!] \rangle \{^{\mathrm{v}(j)}/_x\} \Big)$$

where b is t, if $B = \emptyset$, and f, otherwise, expands the abbreviations into the intended target term by following the semantic rules in Table 5. Branches in *Test*-state are those that carry values (therefore $j \in V$); the substitution $\{^{\mathrm{v}(j)}/_x\}$ replaces the input variable in the continuation process P_j with the corresponding value. Branches in *Read*-state must neither currently carry values ($j \notin V$) nor have accessed the lock after reading values ($j \notin B$).

Lemma 5.3.3 (Factorization) *1. $\mathcal{F}[\![\ ]\!] \circ \mathcal{A}[\![\ ]\!] = \mathcal{C}[\![\ ]\!]$.*
2. $\mathcal{F}[\![\ ]\!]$ is surjective. for all $A \in \mathbb{A}$, $\mathcal{F}[\![A\sigma]\!] = \mathcal{F}[\![A]\!]\sigma$.

Proof:

1. Straightforward induction on the structure of source terms.
2. Straightforward. Neither $\mathcal{A}[\![\ ]\!]$ nor $\mathcal{F}[\![\ ]\!]$ erase names. Free (bound) occurrences of names of terms correspond to free (bound) occurrences in their translations. $\square$

The important property of the factorization is that the semantics of annotated choice (cf. Table 5) precisely mirrors the behavior of the original translations and their derivatives.

Proposition 5.3.4 (Semantic correctness) *$\mathcal{F}$ is a strong bisimulation.*

Proof: By simple induction on the structure of $A \in \mathbb{A}$ and transition induction on $A \to A'$ and $\mathcal{F}[\![A]\!] \to T'$, where it suffices to check the induction case for annotated choice constructors (see Appendix A.1). $\square$

Corollary 5.3.5 (Mirroring) *For all $S \in \mathbb{S}$ and $\mathcal{C}[\![S]\!] \to^* T \in \mathbb{T}$, there is $A \in \mathbb{A}$ such that ($\mathcal{A}[\![S]\!] \to^* A$ and) $\mathcal{F}[\![A]\!] = T$.*

With Proposition 5.3.4 we prove the correctness of $\mathcal{C}[\![\ ]\!]$ by proving the correctness of $\mathcal{A}[\![\ ]\!]$. This is a considerably simpler task, since the $\mathbb{A}$-annotations in the target of $\mathcal{A}[\![\ ]\!]$ provide more structure, in particular concerning partially committed derivatives, which is heavily exploited in §5.4.

5.4 Decoding derivatives of $\mathcal{A}$-translations

We want to construct a coupled simulation (cf. Definition 2.4.1) between source terms and abbreviated target terms by mapping the latter back to the former. Since derivatives of target terms may correspond to source-level choices in a partially committed intermediate state, there are two natural strategies for decoding. The decoding functions $\mathcal{U}_\flat[\![\ ]\!]$ and $\mathcal{U}_\sharp[\![\ ]\!]$

$$\mathbb{S} \underset{\mathcal{U}_\flat[\![\ ]\!]}{\overset{\mathcal{U}_\sharp[\![\ ]\!]}{\leftleftarrows}} \mathbb{A}$$

map partially committed annotated terms in $\mathbb{A}$ back to source terms in $\mathbb{S}$ which are either

- the *least* possible committed (*resetting* decoding $\mathcal{U}_\flat$), or
- the *most* possible committed (*committing* decoding $\mathcal{U}_\sharp$).

The functions $\mathcal{U}_\flat[\![\ ]\!], \mathcal{U}_\sharp[\![\ ]\!] : \mathbb{A} \to \mathbb{S}$ act homomorphically on every constructor but annotated choice according to the scheme in §4. For the latter, we distinguish between choices that are initial, committed, or partial.

For non-partial choices, the two decoding functions have the same definition (read $\mathcal{U}[\![\ ]\!]$ as either $\mathcal{U}_\flat[\![\ ]\!]$ or $\mathcal{U}_\sharp[\![\ ]\!]$): initial choice ($V = \emptyset = B$) is mapped to its bare counterpart,

$$\begin{array}{lrcl}
\textit{initial}: & \mathcal{U}\left[\!\!\left[\ \big(\sum_{j\in J} R_j\big)^{\emptyset}_{\emptyset}\ \right]\!\!\right] & \stackrel{\text{def}}{=} & \sum_{j\in J} \mathcal{U}[\![\, R_j \,]\!] \\
\textit{committed}: & \mathcal{U}\left[\!\!\left[\ \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B\neq\emptyset}\ \right]\!\!\right] & \stackrel{\text{def}}{=} & \prod_{j\in V} \overline{y_j}\, \mathrm{v}_j
\end{array}$$

while committed choice ($B \neq \emptyset$) is mapped to the parallel composition of those messages which are currently held by its branches.

For partial choice ($B = \emptyset$ and $V \neq \emptyset$), the two decoding functions act in a different way according to the intuition described above.

Resetting. The aim is to decode an annotated term to its least possible committed source correspondent. Intuitively, this means that we have to reset all of its partial commitments by mapping it to the original choice in parallel with the already consumed messages.

$$\textit{partial}: \quad \mathcal{U}_\flat\left[\!\!\left[\ \big(\sum_{j\in J} R_j\big)^{\mathrm{v}\neq\emptyset}_{\emptyset}\ \right]\!\!\right] \stackrel{\text{def}}{=} \prod_{j\in V} \overline{y_j}\, \mathrm{v}_j \ \Big|\ \sum_{j\in J} \mathcal{U}_\flat[\![\, R_j \,]\!]$$

Committing. The aim is to decode an annotated term to a committed source correspondent. Intuitively, this means that we have to complete *one* of the (possibly several) activated branches that the annotated choice has engaged in. Let $k := \mathsf{take}(V)$ select an arbitrary element of V. Then,

$$\textit{partial}: \qquad \mathcal{U}_\sharp\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)_{\emptyset}^{\mathrm{v}\neq\emptyset} \,\Big]\!\!\Big] \;\stackrel{\text{def}}{=}\; \prod_{j\in V-k} \overline{y_j}\,\mathrm{v}_j \;\mid\; \mathcal{U}_\sharp[\![\, P_k \,]\!]\{^{\mathrm{v}_k}\!/_x\}$$

maps to the source-level commitment to the selected branch.

Lemma 5.4.1 (Decoding)

1. *Let $\mathcal{U} \in \{\mathcal{U}_\flat, \mathcal{U}_\sharp\}$. Then $\mathcal{U}[\![\ ]\!] \circ \mathcal{A}[\![\ ]\!] = \mathrm{id}$.*
2. *Both $\mathcal{U}_\flat[\![\ ]\!]$ and $\mathcal{U}_\sharp[\![\ ]\!]$ are surjective.*
3. *For $A \in \mathbb{A}$ and $\mathcal{U} \in \{\mathcal{U}_\flat, \mathcal{U}_\sharp\}$, $\mathcal{U}[\![\, A\sigma \,]\!] = \mathcal{U}[\![\, A \,]\!]\sigma$.*

Proof: Immediate by definition (1,2); straightforward by induction (3). □

In the remainder of this subsection, we prove that $(\mathcal{U}_\flat, \mathcal{U}_\sharp)$ is a coupled simulation on $\mathbb{A}\times\mathbb{S}$ (Proposition 5.4.9). This requires, in particular, that $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp^{-1}$ are weak simulations. We will see that these simulations are eventually progressing. First, we show that $\mathcal{U}_\flat[\![\, A \,]\!]$ *simulates* A (i.e., $A \preccurlyeq \mathcal{U}_\flat[\![\, A \,]\!]$) for all $A \in \mathbb{A}$.

Lemma 5.4.2 *$\mathcal{U}_\flat$ is a weak simulation.*

Proof: The proof is by structural induction on $A \in \mathbb{A}$ and transition induction on $\mathcal{U}_\sharp[\![\, A \,]\!] \rightarrow S'$. By the simplification discussed in Section 5.1, it suffices to regard the case

$$A = \big(\sum_{j\in J} R_j\big)_B^{\mathrm{v}}$$

where, by the operational rules in Table 5, there are three subcases.

case (*committed*) $B \neq \emptyset$: Then, $\mathcal{U}_\flat[\![\, A \,]\!] = \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j$.

Since we know by Lemma 5.2.2 that $\big(\sum_{j\in J} R_j\big)_B^{\mathrm{v}} \gtrsim \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j$,

we have that, in this case, $\mathcal{U}_\flat[\![\, A \,]\!]$ is even bisimilar to A.

case (*initial*) $V = \emptyset = B$: Then, $\mathcal{U}_\flat[\![\, A \,]\!] = \sum_{j\in J} \mathcal{U}_\flat[\![\, R_j \,]\!]$

and there is only one type of transitions possible. Note that we do not have to directly simulate input transitions for simulation in the asynchronous π-calculus, but we have to care about the simulation in all contexts, including matching messages. Therefore:

case ***READ/OUT/COM*****:** For $k \in J \backslash (V \cup B)$, we have

$$\overline{y_k}z \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B} \xrightarrow{\tau} \big(\sum_{j\in J} R_j\big)^{\mathrm{v}+(k\mapsto z)}_{B} \qquad \text{and}$$

$$\mathcal{U}_\flat\Big[\!\!\Big[\, \overline{y_k}z \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B} \Big]\!\!\Big] \;=\; \overline{y_k}z \mid \sum_{j\in J} R_j \mid \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j \;\equiv\; \mathcal{U}_\flat\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}+(k\mapsto z)}_{B} \Big]\!\!\Big]$$

immediately concludes the proof by an empty weakly simulating sequence.

case (*partial*) $B = \emptyset \neq V$ **:** Let $k = \mathsf{take}(\mathrm{v})$. There are two subcases:

case ***READ/OUT/COM*****:** (completely analogous to previous case).

case ***COMMIT*****:** For $k \in V$, we have

$$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\emptyset} \xrightarrow{\tau} P_k\sigma_k \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{\{k\}}$$

and via *C-INP/OUT/PAR/COM*, there is the simulating step

$$\begin{aligned} \mathcal{U}_\flat\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\emptyset} \Big]\!\!\Big] \quad &= \quad \sum_{j\in J} R_j \mid \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j \\ &\xrightarrow{\tau} \quad P_k\{{}^{\mathrm{v}_k}\!/\!_z\} \mid \prod_{k\neq j\in V} \overline{y_j}\,\mathrm{v}_j \\ &= \quad \mathcal{U}_\flat\Big[\!\!\Big[\, P_k\sigma_k \mid \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{\{k\}} \Big]\!\!\Big] \end{aligned}$$

which concludes the proof. □

Next, we show that $\mathcal{U}_\sharp[\![\, A \,]\!]$ *is simulated by* A (i.e., $A \succcurlyeq \mathcal{U}_\sharp[\![\, A \,]\!]$) for all $A \in \mathbb{A}$.

Lemma 5.4.3 $\mathcal{U}_\sharp^{-1}$ *is a weak simulation.*

Proof: The proof is by structural induction on $A \in \mathbb{A}$ and transition induction on $\mathcal{U}_\sharp[\![\, A \,]\!] \rightarrow S'$. By the simplification discussed in Section 5.1, it suffices to regard the case

$$A = \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B}$$

where, by definition of $\mathcal{U}_\sharp$, there are three subcases.

case (*initial*) $V = \emptyset = B$ **:** Then $\mathcal{U}_\sharp[\![\, A \,]\!] = \sum_{j\in J} y_j(x).\mathcal{U}_\sharp[\![\, P_j \,]\!]$

where, according to the rule in Table 3, there is only one subcase for generating transitions: *C-INP*. For $k \in J$, we have

$$\sum_{j\in J} y_j(x).\mathcal{U}_\sharp[\![\, P_j \,]\!] \xrightarrow{y_k z} \mathcal{U}_\sharp[\![\, P_k \,]\!]\{{}^z\!/\!_x\}$$

and there is always a weakly simulating sequence by *READ* and *COMMIT*

$$A = \big(\sum_{j\in J} R_j\big)^{\emptyset}_{\emptyset} \xrightarrow{y_k z} \big(\sum_{j\in J} R_j\big)^{(k\mapsto z)}_{\emptyset} \xrightarrow{\tau} P_k\sigma_k \mid \big(\sum_{j\in J} R_j\big)^{\emptyset}_{\{k\}} =: A'$$

where $\mathcal{U}_\sharp[\![\, A' \,]\!] = \mathcal{U}_\sharp[\![\, P_k \,]\!]\{{}^z\!/\!_x\}$ holds.

case (*committed*) $B \neq \emptyset$ **:** Then, $\mathcal{U}_\sharp[\![A]\!] = \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j$.

case *OUT*: For $k \in V$, the transitions

$$\mathcal{U}_\sharp[\![A]\!] \xrightarrow{\overline{y}\mathrm{v}_k} \prod_{j\in V-k} \overline{y_j}\,\mathrm{v}_j =: S_k$$

can be simulated by

$$A \xrightarrow{\tau} \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{B+k} \;\big|\; \overline{y_k}\,\mathrm{v}_k \xrightarrow{\overline{y}\mathrm{v}_k} \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{B+k} =: A_k$$

such that $S_k = \mathcal{U}_\sharp[\![A_k]\!]$.

case (*partial*) $B = \emptyset \neq V$ **:** Let $k = \mathsf{take}(\mathrm{v})$. Then

$$\mathcal{U}_\sharp[\![A]\!] = \prod_{j\in V-k} \overline{y_j}\,\mathrm{v}_j \;\big|\; \mathcal{U}_\sharp[\![P_k]\!]\sigma_k =: S'.$$

By $\quad A \xrightarrow{\tau} \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{\{k\}} \;\big|\; P_k\sigma_k =: A' \quad$ such that $\quad S' = \mathcal{U}_\sharp[\![A']\!]$
we can always take an internal step in order to fully commit A. Note that A' is fully committed since, as a guarded subterm, P_k is fully committed by definition of $\mathbb{A}$. The observation that $S' = \mathcal{U}_\sharp[\![A']\!]$ already concludes the proof since we may reduce it to the case of full terms. □

In addition, the weak simulations $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp^{-1}$ satisfy further useful properties (cf. §5.8).

Lemma 5.4.4

1. *$\mathcal{U}_\flat$ is strict, $\mathcal{U}_\sharp^{-1}$ is progressing.*
2. *Both $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp^{-1}$ are eventually progressing.*

Proof: Both the strictness of $\mathcal{U}_\flat$ and the progressiveness of $\mathcal{U}_\sharp^{-1}$ follow from previous proofs: For $\mathcal{U}_\flat$, proof 5.4.2 shows that some τ-steps of A may be simulated trivially by $\mathcal{U}_\flat[\![A]\!]$:

$$A \xrightarrow{\tau} A' \qquad \text{implies} \qquad \mathcal{U}_\flat[\![A]\!] \xrightarrow{\widehat{\tau}} \mathcal{U}_\flat[\![A']\!]$$

For $\mathcal{U}_\sharp^{-1}$, proof 5.4.3 shows that no τ-step of $\mathcal{U}_\sharp[\![A]\!]$ may be suppressed by A:

$$\mathcal{U}_\sharp[\![A]\!] \xrightarrow{\tau} \mathcal{U}_\sharp[\![A']\!] \qquad \text{implies} \qquad A \stackrel{\tau}{\Longrightarrow} A'$$

Immediately, we get that $\mathcal{U}_\sharp^{-1}$ is eventually progressing.

However, we have not yet proven that also $\mathcal{U}_\flat$ is eventually progressing. We proceed by analyzing τ-sequences starting from an arbitrary $A \in \mathbb{A}$.

$$A = A_0 \xrightarrow{\tau} A_1 \xrightarrow{\tau} A_2 \xrightarrow{\tau} \cdots$$

Since we know that $\mathcal{U}_\flat$ is strict, we have that

$$A_i \xrightarrow{\tau} A_{i+1} \qquad \text{implies} \qquad \mathcal{U}_\flat[\![\, A_i \,]\!] \xrightarrow{\hat{\tau}} \mathcal{U}_\flat[\![\, A_{i+1} \,]\!],$$

so we only have to argue that there is an upper bound k_A for the number of steps where the τ-step may be simulated trivially such that for $n > k_A$:

$$A = A_0 \xrightarrow{\tau} A_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} A_n \qquad \text{implies} \qquad \mathcal{U}_\flat[\![\, A \,]\!] \xrightarrow{\tau}{}^{+} \mathcal{U}_\flat[\![\, A_n \,]\!].$$

For arbitrary steps

$$A_i \xrightarrow{\tau} A_{i+1}$$

we may distinguish four different cases, according to which combination of rules have been applied in the inference. We omit merely structural rules and mention only the essential rules. The following analysis resembles the proof of Lemma 5.4.2. The difference is that, here, we are not interested in visible steps; instead, we take a closer look at the simulations of τ-steps.

case *COM/CLOSE/∗-INP* Here, no choice operator is involved. The transition is caused by a subterm which may be regarded as a target term. The decoding $\mathcal{U}_\flat[\![\ ]\!]$ reproduces this part homomorphically at the source level, wherefore we have

$$\mathcal{U}_\flat[\![\, A_i \,]\!] \xrightarrow{\tau} \mathcal{U}_\flat[\![\, A_{i+1} \,]\!]$$

case *COM/CLOSE/READ* An occurrence of a choice operator is involved. By inspection of rule *READ* and the definition of $\mathcal{U}_\flat[\![\ ]\!]$, we have

$$\mathcal{U}_\flat[\![\, A_i \,]\!] \equiv \mathcal{U}_\flat[\![\, A_{i+1} \,]\!]$$

case *COMMIT* The transition is coming from a partially committed occurrence of annotated choice. Here, we have,

$$\mathcal{U}_\flat[\![\, A_i \,]\!] \xrightarrow{\tau} \mathcal{U}_\flat[\![\, A_{i+1} \,]\!]$$

case *ABORT* Similar to the case before, except that there is no need to perform a τ-step at the source-level.

$$\mathcal{U}_\flat[\![\, A_i \,]\!] \equiv \mathcal{U}_\flat[\![\, A_{i+1} \,]\!]$$

Now, if we look at the above τ-sequence between A_0 and A_n, we have to argue that the second and fourth case cannot happen unboundedly often and, by that, prevent the first and third case. In fact, if A contains no choices, only the first case applies, concluding the proof.

If A contains choices, we may count the number of all branches in *Read*- or *Test*-state of all (unguarded) occurrences of choice, since exactly those may give rise to the sequence of τ-steps. In a partial/initial choice, the number of its branches in *Read*-state, determined by $|J \backslash (V \cup B)|$, tells how many subsequent applications of the second case (*READ*) might be possible, before a *COMMIT*-step has to be derived. In a committed choice, the number of its branches in *Test*-state (provided by $|V|$) yields the number of possible subsequent applications of the forth case (*ABORT*); afterwards, no more τ-steps are generated from this choice, since it is strongly bisimilar to $\mathbf{0}$.

Since each term A may only contain a finite number of unguarded occurrences of choice, k_A is determined as the sum of all of their branches in either *Read*- or *Test*-state. $\square$

Apart from the simulation proofs for the components, a coupled simulation also requires two coupling properties—the existence of internal transition sequences connecting the simulation relations on both sides (Definition 2.4.1). For the proofs of these two properties, which we carry out by induction on the structure of terms, we only exploit the annotations of choices in A to derive the required internal transitions. We start by stating a useful fact.

Fact 5.4.5 *Let $A \in \mathbb{A}$ be fully committed. Then $\mathcal{U}_\sharp\llbracket A \rrbracket = \mathcal{U}_\flat\llbracket A \rrbracket$.*

Full $\mathbb{A}$-terms trivially satisfy coupling properties, since the two decodings coincide. For partially committed terms, this does not hold. There, we have to derive nontrivial internal transitions. Since choice may also occur guarded in a term, we might have to deal with transitions under prefixes, which are forbidden in the operational semantics. Therefore, we must restrict guarded occurrences of choice to being non-partial (e.g. initial)—which is exactly what is guaranteed by the definition of $\mathbb{A}$ in Section 5.3.

Since $\mathcal{U}_\flat\llbracket A \rrbracket \succcurlyeq A$ for all $A \in \mathbb{A}$, the first coupling property requires the existence of transition $\mathcal{U}_\flat\llbracket A \rrbracket \Rightarrow \mathcal{U}_\sharp\llbracket A \rrbracket$ (in $\mathbb{S}$, thus called $\mathbb{S}$-*coupling*).

$$\begin{tikzcd} \mathbb{S}: & \mathcal{U}_\flat\llbracket A \rrbracket \arrow[rr, Rightarrow] \arrow[dr, dotted, no head, "\succcurlyeq"'] & & \mathcal{U}_\sharp\llbracket A \rrbracket \arrow[dl, dotted, no head, "\succcurlyeq"] \\ \mathbb{A}: & & A & \end{tikzcd}$$

Lemma 5.4.6 ($\mathbb{S}$-coupling) *For all $A \in \mathbb{A}$, $\mathcal{U}_\flat\llbracket A \rrbracket \Rightarrow \mathcal{U}_\sharp\llbracket A \rrbracket$.*

Proof: The proof is by structural induction on $A \in \mathbb{A}$. By the simplification discussed in Section 5.1, it suffices to regard the case

$$A = \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B}$$

where, by definition of $\mathcal{U}_\sharp[\![\ ]\!]$, there are three subcases.

case (*initial*) $V = \emptyset = B$ **or (*committed*)** $B \neq \emptyset$ **:**
Immediate by Fact 5.4.5.

case (*partial*) $B = \emptyset \neq V$ **:** Let $k = \mathsf{take}(V)$. Then,

$$\begin{array}{rcl} \mathcal{U}_\flat[\![A]\!] & = & \prod\limits_{j\in V} \overline{y_j}\,\mathrm{v}_j \;\big|\; \sum\limits_{j\in J} \mathcal{U}_\flat[\![R_j]\!] \\ & \xrightarrow{\tau} & \prod\limits_{j\in V-k} \overline{y_j}\,\mathrm{v}_j \;\big|\, \mathcal{U}_\flat[\![P_k]\!]\sigma_k = \prod\limits_{j\in V-k} \overline{y_j}\,\mathrm{v}_j \;\big|\, \mathcal{U}_\sharp[\![P_k]\!]\sigma_k = \mathcal{U}_\sharp[\![A]\!]. \end{array}$$

where $\mathcal{U}_\flat[\![P_k]\!]\sigma_k = \mathcal{U}_\sharp[\![P_k]\!]\sigma_k$ since $P_k\sigma_k$ is fully committed.
There may be several occurrences of partially committed choices in a term A, but, by definition, they only occur unguarded. We may simply collect the corresponding internal steps in either order which leads to $A \Rightarrow A'$. □

Since $A \succcurlyeq \mathcal{U}_\sharp[\![A]\!]$ for all $A \in \mathbb{A}$, the second coupling property addresses $\mathcal{U}_\sharp$-related terms. In this case, it is not as simple as for the $\mathbb{S}$-coupling to denote what coupling means, so we explain it a bit more carefully: For all $A \in \mathbb{A}$, whenever $(A, S) \in \mathcal{U}_\sharp$, i.e., $S = \mathcal{U}_\sharp[\![A]\!]$, there is an internal sequence $A \Rightarrow A'$ (in $\mathbb{A}$, thus called $\mathbb{A}$-*coupling*), such that $(A', S) \in \mathcal{U}_\flat$, i.e., $S = \mathcal{U}_\flat[\![A']\!]$. If we link the two equations for S, we get the coupling requirement $\mathcal{U}_\sharp[\![A]\!] = \mathcal{U}_\flat[\![A']\!]$ for $A \Rightarrow A'$. In the diagram

$$\begin{tikzcd}
\mathbb{S}: & \mathcal{U}_\sharp[\![A]\!] \arrow[d, dotted, dash, "\succcurlyeq"] & {=\mathcal{U}_\sharp[\![A']\!]=} \arrow[dr, dotted, dash, "\preccurlyeq"] & \mathcal{U}_\flat[\![A']\!] \arrow[d, dotted, dash, "\succcurlyeq"] \\
\mathbb{A}: & A \arrow[rr, Rightarrow] & & A'
\end{tikzcd}$$

we also indicate the way we proceed in order to do the proof. The relation on the left is the assumption because $\mathcal{U}_\sharp^{-1}$ is a simulation. The two relations on the right hold if A' is fully committed. The following lemma states that such an A' always exists as a derivative of A and, furthermore, connects the left- and right-hand sides of the diagram.

Lemma 5.4.7 *For all $A \in \mathbb{A}$, there is a fully committing $A \Rightarrow A'$ such that $\mathcal{U}_\sharp[\![A]\!] = \mathcal{U}_\sharp[\![A']\!]$.*

Proof: The proof is by structural induction on $A \in \mathbb{A}$. By the simplification discussed in Section 5.1, it suffices to regard the case

$$A = \big(\sum_{j \in J} R_j \big)^{\mathrm{v}}_{B}$$

where, by definition of $\mathcal{U}_\sharp[\![\;]\!]$, there are three subcases.

case $($***initial***$)$ $V = \emptyset = B$ **or** $($***committed***$)$ $B \neq \emptyset$ **:**

Immediate with $A' \stackrel{\text{def}}{=} A$.

case $($***partial***$)$ $B = \emptyset \neq V$ **:** Let $k = \mathsf{take}(V)$.

Then, with $A \xrightarrow{\tau} \big(\sum_{j \in J} R_j \big)^{\mathrm{v}-k}_{\{k\}} \mid P_k\sigma_k \stackrel{\text{def}}{=} A'$

we have $\mathcal{U}_\sharp[\![A]\!] = \prod_{j \in V-k} \overline{y_j}\,\mathrm{v}_j \mid \mathcal{U}_\sharp[\![P_k]\!]\sigma_k = \mathcal{U}_\sharp[\![A']\!]$.

Note that P_k is fully committed since it was guarded in A and not changed by the transition; thus, A' is fully committed, since branch k has been successfully chosen. $\square$

Lemma 5.4.8 ($\mathbb{A}$-coupling) *For all $A \in \mathbb{A}$, there is $A \Rightarrow A'$ such that $\mathcal{U}_\sharp[\![A]\!] = \mathcal{U}_\flat[\![A']\!]$.*

Proof: Let A' be constructed as in the proof of Lemma 5.4.7. Thus, we know that $\mathcal{U}_\sharp[\![A]\!] = \mathcal{U}_\sharp[\![A']\!]$ and, since A' is fully committed, we also know (Fact 5.4.5) that $\mathcal{U}_\sharp[\![A']\!] = \mathcal{U}_\flat[\![A']\!]$. $\square$

Finally, the main property of the decoding functions $\mathcal{U}_\flat[\![\;]\!]$ and $\mathcal{U}_\sharp[\![\;]\!]$ is:

Proposition 5.4.9 $(\mathcal{U}_\flat,\mathcal{U}_\sharp)$ *is a coupled simulation.*

Proof: By Lemmas 5.4.2 and 5.4.3, we know that $(\mathcal{U}_\flat,\mathcal{U}_\sharp)$ is a mutual simulation. By Lemmas 5.4.6 and 5.4.8, we have the necessary coupling between $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp$. $\square$

Corollary 5.4.10 $(\mathcal{U}_\sharp^{-1},\mathcal{U}_\flat^{-1})$ *is a coupled simulation.*

Proof:

By Fact 2.4.2. $\square$

Before ending this section, we show how the decoding functions may be used to provide a notably sharp operational correctness argument for the $\mathcal{A}$-encoding. Let $\xRightarrow{a_1 \cdots a_n}$ denote the sequence $\xRightarrow{a_1} \cdots \xRightarrow{a_n}$ where the a_i are output-labels. Then, for arbitrary $S \in \mathbb{S}$ and $\mathcal{A}[\![\, S \,]\!] \xRightarrow{a_1 \cdots a_n} A$, we have the following relations:

$$
\begin{tikzcd}
\mathbb{S}: & S \arrow[r, Rightarrow, "a_1 \cdots a_n"] \arrow[d, dotted, dash, bend right, "\preccurlyeq"'] \arrow[d, dotted, dash, bend left, "\succcurlyeq"] & \mathcal{U}_\flat[\![\, A \,]\!] \arrow[r, Rightarrow] \arrow[dr, dotted, dash, "\succcurlyeq"] & \mathcal{U}_\sharp[\![\, A \,]\!] =: S' \arrow[d, dotted, dash, "\succcurlyeq"'] \arrow[dr, dotted, dash, bend right, "\preccurlyeq"'] \arrow[dr, dotted, dash, bend left, "\succcurlyeq"] & \\
\mathbb{A}: & \mathcal{A}[\![\, S \,]\!] \arrow[rr, Rightarrow, "a_1 \cdots a_n"] & & A \arrow[r, Rightarrow] & \mathcal{A}[\![\, S' \,]\!]
\end{tikzcd}
$$

each derivative A of a translation $\mathcal{A}[\![\, S \,]\!]$ represents an intermediate state in some source-level computation evolving from an S-derivative $\mathcal{U}_\flat[\![\, A \,]\!]$ into an S-derivative $S' := \mathcal{U}_\sharp[\![\, A \,]\!]$, as proved by Proposition 5.4.6 ($\mathbb{S}$-coupling). Furthermore, by Proposition 5.4.8 ($\mathbb{A}$-coupling), A can internally evolve into the $\mathcal{A}$-translation of S'. In arbitrary contexts within $\mathbb{S}$, the derivation trace of S corresponds to the derivation trace of $\mathcal{A}[\![\, S \,]\!]$, since $\mathcal{U}_\flat$ is a weak simulation containing $(\mathcal{A}[\![\, S \,]\!], S)$. Finally, $\mathcal{U}_\flat[\![\, A \,]\!]$ and $\mathcal{U}_\sharp[\![\, A \,]\!]$ coincide in the case that A is not partial.

The results of this subsection provide a very tight correspondence between source terms S and their annotated translations $\mathcal{A}[\![\, S \,]\!]$. It may be written as $S \rightleftarrows \mathcal{A}[\![\, S \,]\!]$ for every $S \in \mathbb{S}$, since every pair $(S, \mathcal{A}[\![\, S \,]\!])$ is contained in both components of the coupled simulation $(\mathcal{U}_\sharp^{-1}, \mathcal{U}_\flat^{-1})$. However, we are not primarily interested in properties of $\mathcal{A}[\![\ ]\!]$, but in properties of $\mathcal{C}[\![\ ]\!]$ and $\mathcal{C}_{\lesssim}[\![\ ]\!]$. Those results are assembled in §5.6 after we have shown rigorously in the next subsection that the use of booleans in the target language $\mathbb{T}$ is of mere notational convenience.

5.5 Expanding derivatives of $\mathcal{C}$-translations

Another advantage of abbreviated target terms is that we can use their structure to prove that boolean messages and `test`-expressions are correctly expanded into terms of the language $\mathbb{P}$. More precisely, the above Corollary 5.3.5 allows us to quantify over all derivatives T of $\mathcal{C}$-translations by simply quantifying over all terms A in $\mathbb{A}$, and assuming that $T = \mathcal{F}[\![\, A \,]\!]$.

We start with an operational correspondence result for the $\mathcal{B}$-encoding of derivatives of $\mathcal{C}$-translations. It makes explicit the number of additional τ-steps that are needed in order to perform the exchange of 2-adic messages via the zip-lock encoding, and the subsequent triggering action, where the void name is exchanged.

Lemma 5.5.1 (Operational correspondence for $\mathcal{B}[\![\ ]\!]$)
Let $T := \mathcal{F}[\![\, A \,]\!]$ for some $A \in \mathbb{A}$.

1. *Let $\mu \neq \tau$. Then $T \xrightarrow{\mu} T'$ iff $\mathcal{B}[\![\, T \,]\!] \xrightarrow{\mu} \mathcal{B}[\![\, T' \,]\!]$.*

2. *If* $T \xrightarrow{\tau} T'$*, then there is*
   either $\mathcal{B}[\![\, T \,]\!] \xrightarrow{\tau} \mathcal{B}[\![\, T' \,]\!]$ *or* $\mathcal{B}[\![\, T \,]\!] \xrightarrow{\tau}{}^{6} \equiv \mathcal{B}[\![\, T' \,]\!]$*.*
3. *If* $\mathcal{B}[\![\, T \,]\!] \xrightarrow{\tau} P$*, then there is* $T \xrightarrow{\tau} T'$ *such that*
   either $P = \mathcal{B}[\![\, T' \,]\!]$ *or* $P \xrightarrow{\tau}{}^{5} \equiv \mathcal{B}[\![\, T' \,]\!]$*.*

Proof: By transition induction on $\mathcal{F}[\![\, A \,]\!] \xrightarrow{\mu} T'$ and $(\mathcal{B} \circ \mathcal{F})[\![\, A \,]\!] \xrightarrow{\mu} P$ (see §A.2). $\square$

Note that part *3* of the operational correspondence lemma witnesses the promptness of the $\mathcal{B}$-encoding: the first low-level interaction in

$$\mathcal{B}[\![\, T \,]\!] \xrightarrow{\tau} P \xrightarrow{\tau}{}^{5} \equiv \mathcal{B}[\![\, T' \,]\!]$$

always corresponds to some $\mathbb{T}$-level interaction.

According to the proof, the rather constrained behavior of interactions along the channels u, v, w, t, f (as introduced by the $\mathcal{B}$-encoding) is due to its temporarily determinate character. As soon as the first private interaction has taken place (via l according to the encoding functions), the following five interactions are determined since any of the above private channels is used uniquely and is restricted to the outside world—no other actions may prevent those five from taking place. This leads us to a strengthening of cases *2* and *3* of the operational correspondence lemma, so for the intermediate states P' after the first interaction and before reaching the expansion of the successor state:

$$\mathcal{B}[\![\, T \,]\!] \xrightarrow{\tau} P \;\Rightarrow\; P' \;\Rightarrow\equiv\; \mathcal{B}[\![\, T' \,]\!] \quad \text{implies} \quad P' \gtrsim \mathcal{B}[\![\, T' \,]\!].$$

Lemma 5.5.2 *Let* $T := \mathcal{F}[\![\, A \,]\!]$ *for some* $A \in \mathbb{A}$*.*

2. *If* $T \xrightarrow{\tau} T'$*, then there is* $\mathcal{B}[\![\, T \,]\!] \xrightarrow{\tau} P$ *with* $P \gtrsim \mathcal{B}[\![\, T' \,]\!]$*.*
3. *If* $\mathcal{B}[\![\, T \,]\!] \xrightarrow{\tau} P$*, then there is* $T \xrightarrow{\tau} T'$ *and* $P \gtrsim \mathcal{B}[\![\, T' \,]\!]$*.*

Proof: By looking closer at the proof for operational correspondence in Appendix A.2; all the details are already there. $\square$

In order to carry over this result to the relation between test-expressions and their $\mathcal{B}$-translations, we define an extension of the relation $\mathcal{B}$ by including the required pairs for intermediate derivatives P' of $\mathcal{B}[\![\, T \,]\!]$.

$$\begin{aligned} \mathcal{B}^{\rightarrow} \overset{\text{def}}{=} \mathcal{B} \cup \{ \quad & (T', P') \in \mathbb{P}^{\mathsf{test}} \times \mathbb{P} \;\big|\; \text{there is } T = \mathcal{F}[\![\, A \,]\!] \text{ for } A \in \mathbb{A} \\ & \text{such that } T \xrightarrow{\tau} T' \text{ and } \mathcal{B}[\![\, T \,]\!] \xrightarrow{\tau}{}^{+} P' \;\Rightarrow\equiv\; \mathcal{B}[\![\, T' \,]\!] \;\} \end{aligned}$$

Lemma 5.5.3 $\mathcal{B}^{\rightarrow}$ *is an expansion.*

Proof: Immediate by the operational correspondence (Lemma 5.5.1) and the expansion property for all intermediate states P'. $\square$

Fact 5.5.4 *For all $T = \mathcal{F}[\![\, A \,]\!]$ for $A \in \mathbb{A}$: $(T, \mathcal{B}[\![\, T \,]\!]) \in \mathcal{B}^{\rightarrow}$.*

Corollary 5.5.5 *Let $A \in \mathbb{A}$ and $S \in \mathbb{S}$.*

1. *For all $T := \mathcal{F}[\![\, A \,]\!]$: $T \lesssim \mathcal{B}[\![\, T \,]\!]$.*
2. *For all $\mathcal{C}[\![\, S \,]\!] \xrightarrow{\tau}{}^{*} T$: $T \lesssim \mathcal{B}[\![\, T \,]\!]$.*

Proof: (1) is a consequence from Fact 5.5.4 and Lemma 5.5.3. (2) is then a direct consequence of (1) and Corollary 5.3.5. $\square$

Lemma 5.5.6 *$\mathcal{B}^{\rightarrow}$ is progressing; $(\mathcal{B}^{\rightarrow})^{-1}$ is eventually progressing.*

Proof: Directly from the operational correspondence Lemma 5.5.1. The eventually progressing property for $(\mathcal{B}^{\rightarrow})^{-1}$ comes with an upper bound for trivially simulating τ-steps as determined by the (finite) number of 'active' test-expressions, multiplied with 5 as the worst case that all of the active test-expressions have just done the first step. $\square$

5.6 Main result

In this section, we establish a coupled simulation (cf. Definition 2.4.1) between source terms and their $\mathcal{C}_{\lesssim}$-translations by exploiting the results for the $\mathcal{A}$-encoding via the decodings $\mathcal{U}_\flat[\![\ ]\!]$ and $\mathcal{U}_\sharp[\![\ ]\!]$. Reasoning about the annotated versions of choice allowed us to use their high-level structure for the decoding functions.

We argued that we could safely concentrate on the annotated language $\mathbb{A}$, since $\mathcal{F}[\![\ ]\!]$ flattens abbreviation terms correctly (up to $\sim$) into terms of $\mathbb{P}^{\mathsf{test}}$, whereas $\mathcal{B}[\![\ ]\!]$ expands test-expressions correctly (up to $\lesssim$) into terms of $\mathbb{P}$. In order to combine those ideas, let the simulations $\mathfrak{C}$ (completeness) and $\mathfrak{S}^{-1}$ (soundness) be defined by

$$\mathfrak{C} \stackrel{\text{def}}{=} \mathcal{U}_\sharp^{-1}\mathcal{F}\,\mathcal{B}^{\rightarrow} \qquad \text{and} \qquad \mathfrak{S} \stackrel{\text{def}}{=} \mathcal{U}_\flat^{-1}\mathcal{F}\,\mathcal{B}^{\rightarrow}$$

according to the diagram

$$\begin{tikzcd}
\mathbb{P}^{\Sigma} \arrow[d, shift right, Rightarrow, "\mathfrak{C}"'] \arrow[d, shift left, Rightarrow, "\mathfrak{S}"] & \mathbb{P}^{(\Sigma)} \arrow[l, dotted, shift right, "\mathcal{U}_\sharp"'] \arrow[l, dotted, shift left, "\mathcal{U}_\flat"] \arrow[d, dotted, "\mathcal{F}"] \\
\mathbb{P} & \mathbb{P}^{\mathsf{test}} \arrow[l, dotted, "\mathcal{B}^{\rightarrow}"]
\end{tikzcd}$$

where the relations $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp$ are only defined on the subset $\mathbb{A}$ of $\mathbb{P}^{(\Sigma)}$. The results for annotated terms carry over smoothly to the expanded versions.

Theorem 5.6.1 *($\mathfrak{C}, \mathfrak{S}$) is a coupled simulation.*

Proof: By Corollary 5.4.10, Proposition 5.3.4, and Lemma 2.5.3 twice. □

Observe that $\mathfrak{C}$ is constructed from the committing decoding $\mathcal{U}_\sharp[\![\ ]\!]$, so derivatives of target terms are at most as committed as their $\mathfrak{C}$-related source terms. Analogously, $\mathfrak{S}$ is constructed from the resetting decoding $\mathcal{U}_\flat[\![\ ]\!]$, so derivatives of target terms are at least as committed as their $\mathfrak{S}$-related source terms.

By construction, the relations $\mathfrak{C}$ and $\mathfrak{S}$ are big enough to contain all source and target terms and, in particular, to relate all source terms and their $\mathcal{C}_{\lesssim}$-translations.

Lemma 5.6.2 *For all $S \in \mathbb{S}$: $(S, \mathcal{C}_{\lesssim}[\![\, S \,]\!]) \in \mathfrak{C} \cap \mathfrak{S}$.*

Proof: By the syntactic adequacy lemmas (5.4.1 and 5.3.3), we know that, for all $S \in \mathbb{S}$, the translations $\mathcal{A}[\![\, S \,]\!]$ and $\mathcal{C}[\![\, S \,]\!] = (\mathcal{F} \circ \mathcal{A})[\![\, S \,]\!]$ yield the witnesses for $(S, \mathcal{C}_{\lesssim}[\![\, S \,]\!])$ being contained in both $\mathfrak{C}$ and $\mathfrak{S}$. □

Thus, the $\mathcal{C}_{\lesssim}$-encoding is operationally correct, in the sense that every source term is simulated by its translation (completeness via $\mathfrak{C}$) and also itself simulates its translation (soundness via $\mathfrak{S}$). Moreover, the result is much stronger since the simulations are coupled.

Theorem 5.6.3 (Correctness of $\mathcal{C}_{\lesssim}$) *For all $S \in \mathbb{S}$: $S \rightleftarrows \mathcal{C}_{\lesssim}[\![\, S \,]\!]$.*

Proof: By Theorem 5.6.1 and Lemma 5.6.2. □

Corollary 5.6.4 (Full abstraction) *For all $S_1, S_2 \in \mathbb{S}$: $S_1 \rightleftarrows S_2$ iff $\mathcal{C}_{\lesssim}[\![\, S_1 \,]\!] \rightleftarrows \mathcal{C}_{\lesssim}[\![\, S_2 \,]\!]$.*

Proof: By transitivity. □

Note that the $\mathcal{C}_{\lesssim}$-encoding is not fully abstract up to weak bisimulation, as proven in §4.3.

5.7 Correctness of the divergent protocol

The proof for the divergent choice encoding follows the outline of Sections 5.2 to 5.6. In contrast to the divergence-free encoding, the $\mathcal{D}_{\lesssim}$-encoding *is* correct up to weak bisimulation. The overall proof is simpler since the proof obligations for weak bisimulation require less work than those of coupled simulation. We sketch the full proof here by carefully defining just the main ingredient of the factorization and decoding diagram:

$$
\begin{tikzcd}
\mathbb{S} \arrow[r, shift right, "\mathcal{A}[\![\ ]\!]"'] \arrow[d, "\mathcal{D}_{\lesssim}[\![\ ]\!]"'] \arrow[dr, dotted] & \mathbb{A} \arrow[l, shift right, "\mathcal{U}[\![\ ]\!]"'] \arrow[d, "\mathcal{F}[\![\ ]\!]"] \\
\mathbb{P} & \mathbb{T} \arrow[l, "\mathcal{B}[\![\ ]\!]"]
\end{tikzcd}
$$

The annotated intermediate language $\mathbb{A}$ that we use here is similar to the one for the $\mathcal{C}$-encoding. Of course, the annotations and the operational semantics have to be different in the case of the $\mathcal{D}$-encoding, since they are now expected to model different behavior. Also, since we expect source terms and translations to be bisimilar, we only need a single decoding function.

Annotated divergent choice. Since many (replicated) copies of the same branch may be activated at the same time, we enhance the annotation v to map indices to multisets of values $\mathbb{N}^{\mathbf{V}}$. The information whether a choice is resolved can only be inferred from the value of the lock. In case it is held by an activated branch, the choice is not (yet) resolved, since this branch may still decide to undo its activity.

Let J be some indexing set. Let $\mathrm{v} : J \rightharpoonup \mathbb{N}^{\mathbf{V}}$ be a partial function mapping indices to multisets of values and let $\mathrm{v} + (k, z)$ and $\mathrm{v} - (k, z)$ denote appropriate extensions and removals of single index-value pairs. Let $\mathsf{b} \in \{\mathsf{t}, \mathsf{f}\} \cup \{ (k, z) \in J \times \mathbf{V} \mid z \in \mathrm{v}_k \}$ denote either the state of the lock or a single index-value pair. The annotated divergent choice

$$\Big(\sum_{j \in J} R_j \Big)_{\mathsf{b}}^{\mathrm{v}}$$

is given the operational semantics in Table 6.

Lemma 5.7.1 (Inertness) $\big(\sum_{j \in J} R_j \big)_{\mathsf{f}}^{\mathrm{v}} \approx \prod_{j \in V} \overline{y_j}\, \mathrm{v}_j$.

Factorization. The *annotation* translation is almost the same as in Section 5.3. Here, we have to initialize the lock-information with t.

$$\mathcal{A}\Big[\!\!\Big[\sum_{j \in J} R_j \Big]\!\!\Big] \stackrel{\text{def}}{=} \Big(\sum_{j \in J} \mathcal{A}[\![R_j]\!] \Big)_{\mathsf{t}}^{\emptyset}$$

READ:	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\mathsf{b}}$	$\xrightarrow{y_k z}$	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}+(k,z)}_{\mathsf{b}}$	*if* $k \in J$
TRY:	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\mathsf{t}}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{(k,z)}$	*if* $z \in \mathrm{v}_k$
COMMIT:	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{(k,z)}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}-(k,z)}_{\mathsf{f}} \mid P_k\{^z\!/_x\}$	
UNDO:	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{(k,z)}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}-(k,z)}_{\mathsf{t}} \mid \overline{y_k}z$	
ABORT:	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\mathsf{f}}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}-(k,z)}_{\mathsf{f}} \mid \overline{y_k}z$	*if* $z \in \mathrm{v}_k$

Table 6: Transition semantics for annotated divergent choice

The *flattening* of annotated divergent choice into $\mathbb{T}$ is

$$\mathcal{F}\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B} \,\Big]\!\!\Big] \stackrel{\text{def}}{=} (l)\Big(T \;\Big|\; \prod_{j\in J} \,!\,\mathit{Read}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle \;\Big|\; \prod_{(j,z)\in V-\mathsf{b}} \mathit{Test}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle\{^z\!/_x\} \Big)$$

$$\text{where} \quad T \stackrel{\text{def}}{=} \begin{cases} \big(\, \mathit{Commit}_l\langle\, R_k \,\rangle \oplus \mathit{Undo}_l\langle\, R_k \,\rangle \,\big)\{^z\!/_x\} & \text{if } \mathsf{b} = (k,z) \\ \bar{l}\mathsf{b} & \text{otherwise} \end{cases}$$

Proposition 5.7.2 *$\mathcal{F}$ is a strong bisimulation.*

Decoding. The decoding function also takes care of 'hesitating' branches.

$$\mathcal{U}\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\mathsf{b}} \,\Big]\!\!\Big] \stackrel{\text{def}}{=} \prod_{j\in V} \overline{y_j}\,\mathrm{v}_j \;\Big|\; S$$

$$\text{where} \quad S \stackrel{\text{def}}{=} \begin{cases} \sum_{j\in J} \mathcal{U}[\![\, R_j \,]\!] & \text{if } \mathsf{b} = \mathsf{t} \text{ or } \mathsf{b} = (k,z) \\ \mathbf{0} & \text{if } \mathsf{b} = \mathsf{f} \end{cases}$$

Proposition 5.7.3 *$\mathcal{U}$ is a weak bisimulation.*

Theorem 5.7.4 *$\mathcal{U}^{-1}\mathcal{F}$ is a weak bisimulation.*

Theorem 5.7.5 (Correctness of $\mathcal{D}$) *For all $S \in \mathbb{S}$: $S \approx \mathcal{D}[\![\, S \,]\!]$.*

Corollary 5.7.6 (Full abstraction) *For all* $S_1, S_2 \in \mathbb{S} : S_1 \approx S_2$ *iff* $\mathcal{D}[\![S_1]\!] \approx \mathcal{D}[\![S_2]\!]$.

Every annotated choice term with at least one branch holding some value, has τ-loops, i.e., divergent computations. For example,

$$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\mathtt{t}} \;\xrightarrow{\tau}\; \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{(k,z)} \;\xrightarrow{\tau}\; \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-(k,z)}_{\mathtt{t}} \mid \overline{y_k}z \;\xrightarrow{\tau}\; \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\mathtt{t}}$$

where $z \in \mathrm{v}_k$, is infinitely often trying, undoing, reading, ...

Finally, if we define $\mathcal{D}_{\lesssim}[\![\ ]\!]$ as $\mathcal{B}[\![\ ]\!] \circ \mathcal{D}[\![\ ]\!]$, then by proving that $\mathcal{U}^{-1}\mathcal{F}\mathcal{B}^{\rightarrow}$ is a weak bisimulation, we also get that $\mathcal{D}_{\lesssim}[\![\ ]\!]$ is correct and fully abstract with respect to weak bisimulation.

5.8 Divergence

In §4, we have claimed that our two choice encodings differ in their divergence-behavior. Our results in the previous subsections do not yet provide any rigorous justification of this, since the definition of weak simulation ignores divergence, as we discussed in Section 2.3.

As explained in §2.3, an encoding $[\![\ ]\!]$ is called divergence-free if it does not add divergence to the behavior of source terms. More technically, every infinite τ-sequence of a derivative T' of a translation $[\![S]\!]$ corresponds to some infinite τ-sequence of a derivative S' of S, i.e., $T'\Uparrow$ implies $S'\Uparrow$. Whereas it is simple to show that the $\mathcal{D}_{\lesssim}$-encoding is not divergence-free by giving an example of a τ-loop (which is possible in almost every $\mathcal{D}$-translation for terms containing choices, and carries over to their $\mathcal{B}$-expansions), as we did in the previous section, our claim that the $\mathcal{C}$-encoding is divergence-free requires a proof (cf. Theorem 5.8.2).

We are going to prove the divergence-freedom by establishing an eventually progressing simulation between derivatives of translations and source terms; by this simulation, we capture any reachable state of some translation and are guaranteed that its divergence was not introduced by the encoding, but already present at the source-level.

A good candidate for such a simulation is $\mathfrak{S}^{-1}$ since it relates all derivatives of $\mathcal{C}_{\lesssim}$-translations and is itself composed of three eventually progressing simulations.

Proposition 5.8.1 $\mathfrak{S}^{-1}$ *is eventually progressing.*

Proof: By $\mathfrak{S} \overset{\text{def}}{=} \mathcal{U}_{\flat}^{-1}\mathcal{F}\,\mathcal{B}^{\rightarrow}$, we have $\mathfrak{S}^{-1} = (\mathcal{B}^{\rightarrow})^{-1}\mathcal{F}^{-1}\mathcal{U}_{\flat}$. Since all of $(\mathcal{B}^{\rightarrow})^{-1}$, $\mathcal{F}^{-1}$, and $\mathcal{U}_{\flat}$ are eventually progressing, there composition is also eventually progressing, as proved by simply multiplying the upper bounds for the number of trivial simulation steps of the components. $\Box$

Theorem 5.8.2 $\mathcal{C}_{\lesssim}[\![\ ]\!]$ *is divergence-free.*

Proof: Since $(S, \mathcal{C}_{\lesssim}[\![S]\!]) \in \mathfrak{S}^{-1}$ for all $S \in \mathbb{S}$, and $\mathfrak{S}^{-1}$ is an eventually progressing simulation by Proposition 5.8.1. $\Box$

6 Conclusions

We have investigated two different encodings of the asynchronous π-calculus with input-guarded choice into its choice-free fragment. Several points deserve to be discussed.

Correctness: For both choice encodings, we provided a framework that allowed us to compare source terms and their translations directly. This enabled us to use a correctness notion that is stronger than the usual full abstraction, which here comes up as a simple corollary. The strength of our results may be compared with the notion of representability in [HY94a, HY94b], where it was left as an open problem whether some form of summation could be behaviorally represented by concurrent combinators. Our divergent encoding (for theoretical questions like representability, divergence is acceptable) provides a first positive answer for the representability of input-guarded choice up to weak asynchronous bisimulation.

Our results also hold in the setting of value-passing CCS with boolean values and test-expressions. However, in the π-calculus, these additional notions can be encoded by a simple name-passing protocol (as shown in this paper). Furthermore, the results (except for the congruence properties) can be generalized to calculi with polyadic communication, full replication, and matching.

Asynchrony: For both encodings, the correctness proofs cannot be built upon standard (i.e., synchronous) notions of simulation. The reason lies in the inherent asynchrony of the implementation algorithm, which arises from the resending of messages (which must not be kept by a branch when the choice has already committed to a competing branch).

Non-promptness: Most examples of encodings into process calculi known in the literature enjoy the simplifying property of being *prompt*, i.e., initial transitions of translations are committing, by corresponding to some particular computation step of their source. Both of our choice encodings fall in the class of non-prompt encodings that, moreover, can not be dealt with by optimization with administrative normal forms.

Partial commitments: With respect to the different results for the two choice encodings, it is crucial to notice that only $\mathcal{C}[\![\ ]\!]$ breaks up the atomicity of committing a choice. The resulting partially committed states are exactly the reason why correctness up to weak bisimulation has to fail, whereas coupled simulation applies successfully.

Divergence: We have not been able to formulate a choice encoding which is divergence-free *and* correct with respect to weak bisimulation. We conjecture that it is impossible.

Decodings: Any operational correctness proof which states that an encoding is *sound* in the sense that each step of a translation is compatible with some source step implicitly uses the idea of mapping back the translation to its source term in order to detect the correspondence. We made this intuition explicit in decoding functions which provide a notation for the proofs that is both compact and intuitive. With prompt encodings, the reconstruction of source terms from target terms is rather simple, since it suffices to deal with literal translations. In contrast, non-prompt encodings require the decoding of *derivatives* of translations.

Annotations: The only way to detect the origin of derivatives of translations is to retrace their derivation histories. As the underlying semantics, one could, for example, use causality-based techniques, but this would introduce extra technical overhead. Instead, we exploit annotated source terms that precisely capture the information that is necessary to perform the backtracking. An intermediate language built from annotated source terms provides the basis for a sound factorization and a proper setting for the definition of decodings.

According to the soundness interpretation of the correctness results for the $\mathcal{A}$-encoding in §5.4, the operational character of coupled simulation $(\mathfrak{C}, \mathfrak{S})$ also induces a sharp characterization the operational soundness of the $\mathcal{C}$-encoding: for all source terms $S \in \mathbb{S}$,

$$\begin{tikzcd}
\mathbb{P}^{\Sigma}: & S \arrow[r, Rightarrow, "a_1 \cdots a_n"] \arrow[d, dotted, no head, bend right, "\mathfrak{C}"'] \arrow[d, dotted, no head, bend left, "\mathfrak{S}"] & S' \arrow[rr, Rightarrow] \arrow[dr, dotted, no head, "\mathfrak{S}"] & & S'' \arrow[dl, dotted, no head, "\mathfrak{C}"'] \arrow[d, dotted, no head, bend right, "\mathfrak{C}"'] \arrow[d, dotted, no head, bend left, "\mathfrak{S}"] \\
\mathbb{P}: & \mathcal{C}_{\lesssim}[\![S]\!] \arrow[rr, Rightarrow, "a_1 \cdots a_n"] & & T \arrow[r, Rightarrow] & \mathcal{C}_{\lesssim}[\![S'']\!]
\end{tikzcd}$$

an arbitrary derivative T of a translation represents an intermediate state in some source-level computation evolving from S' into S''. The correspondence between the derivation traces of target $\mathcal{C}[\![S]\!]$ and source S is guaranteed by $\mathfrak{S}$. The relations $S' \succcurlyeq T \succcurlyeq S''$ hold due to $\mathfrak{S}$ and $\mathfrak{C}$, respectively. If T is not partial, then S' and S'' coincide.

Related work

The $\mathcal{C}$-encoding represents a striking example where weak bisimulation is too strong a criterion to compare process systems. It is similar to the multiway synchronization example of [PS92]. The latter led to the definition of coupled simulation in order to deal with gradual commitments, which do also appear in the $\mathcal{C}$-encoding. Our encodings differ in that they address the implementation of channel-based choice in the context of an underlying medium supplying asynchronous message-passing; they are thus more closely related to the work of Mitchell [Mit86], Knabe [Kna93], and Busi and Gorrieri [BG95].

In [Mit86], a divergent choice encoding in the rather restricted setting of Static CCS was proved correct with respect to an adapted ('weak-must') testing equivalence that accepts divergent implementations ($\tau^{\omega}|P$) of P as valid, but that lacks a powerful (e.g. coinductive) proof technique. Here, we have given two asynchronous choice encodings, one of which is divergence-free, and presented a way to prove them correct using asynchronous simulation techniques. Since the original definition of stably coupled simulation equivalence has been shown to imply testing equivalence, we argue that our correctness result for the $\mathcal{C}$-encoding is powerful, even though it is strictly more permissive than weak bisimulation.

The distributed implementation of mixed guarded channel-based choice by Knabe [Kna93] has not been investigated concerning its functional correctness. The emphasis was more on the question of deadlock-freedom; a proof was sketched in [Kna93]. However, the semantics and implementation of this and other choice operators have been studied within the chemical abstract machine framework for the semantics of Facile by Leth and Thomsen [LT95].

In the CCS-setting of Busi and Gorrieri [BG95], choice is replaced by a lower-level notion of *conflict* that is based on a set of conflict names (and contrasting conames) together with corresponding prefix and restriction operators. An operational semantics keeps track of the set of conflicts that a process must respect as permission for performing actions; an auxiliary *kill*-operator deals with the proper handling of permissions. The idea is that, in a process $P|Q$, "if P performs an action, it propagates its effect to Q by killing its conflicting subagents". Then, the hidden activities that go on in a process like $P + Q$ are modeled explicitly by means of a fully abstract (w.r.t. strong bisimulation) encoding of choice into this calculus. Instead of interpreting $P + Q$ as first resolving the choice, their approach is guided by the idea of *a posteriori* choice which means that both P and Q may start their activities, and the first that manages to complete its action wins, preventing the other from completion. Some similarities and differences between their approach and ours are clear: whereas we use the primitives of a fragment of the source language by only exploiting the concept of asynchronous communication of private names, their approach needs the additional concept of conflict names. In fact, the choice encodings of the current paper follow the same idea of a posteriori choice as in [BG95]; yet, we go a step further. In our case, concurrent branches in a choice may start their activity by consuming matching messages from the environment that afterwards might have to be given back. Technically, their encoding introduces conflict names for each occurrence of $P + Q$ and restricts that name on the parallel composition of the branches P and Q, ours introduce lock-messages that are accessible within restricted scope and perform the propagation of conflicts via internal communication.

The idea of committing steps, i.e., those target steps which directly correspond to a source-level computation step, is comparable to the notion of *principal transition* that has been developed for proving the correctness of a compiler from an Occam-like programming language into an assembler language [Gam91]. However, in this setting principal steps could always be chosen as the initial steps. Committing steps have also been recognized by distinguishing *real* and *administrative* steps in the non-prompt encoding of Facile [Ama94] and the concurrent λ-calculus [ALT95] into the π-calculus. In both settings, however, pre-administrative steps were normalizable, allowing for an optimized prompt encoding, which is not the case in the choice encodings.

Encodings of languages into fragments of themselves have been proposed or investigated by several authors, e.g. by the study of encodings of higher-order-communication into first-order communication within process calculi [San93, Tho93, Ama93], the translation of polyadic into monadic π-calculus [Mil93], the implementation of synchronous via asynchronous message-passing within the choice-free (mini) π-calculus [Bou92], several encodings within a hierarchy of π-calculi with internal mobility [San96], the encoding of the choice-free asynchronous π-calculus into the join-calculus [FG96] (which may be interpreted as a fragment of the π-calculus), and the translation of the choice-free synchronous π-calculus into trios [Par99].

Much more work has been done on the compilation of whole languages into process calculi, exploring both semantics and expressiveness. Examples include translations between the process calculi CSP and CCS [Li83, Mil87], between the join-calculus and the π-calculus [FG96], of λ-calculi into process calculi [Mil92, San94b, Lav93, San94a, Tho95, San94b, San95a, ALT95, Ode95b, Nie96], data types and other sequential programming constructs [Mil89, Mil93, Wal91b, Ode95a], from object-oriented languages into process calculi [Vaa90, Wal91a, Wal95, Wal93, Jon93, Wal94, PT95], from logic programming languages into the π-calculus [Ros93, Li94], and from concurrent constraint languages into the π-calculus [Smo94, VP96].

The formalization of compilers for concurrent languages has also motivated the study of encodings, e.g. for Occam [Gam91], Facile [Ama94], Urlang [Gla94], and Pict [PT99]. Further interesting examples report on translations between various notions of rewrite systems [OR94, But94], addressing both semantics and implementation issues.

Future work

The observation that the completeness simulation $\mathfrak{C}$ and the soundness simulation $\mathfrak{S}$ are progressing and strict, respectively, and both eventually progressing, suggests the study of enhancements of coupled simulation equivalence. The coupled simulation $(\mathfrak{C}, \mathfrak{S})$ does not constitute an expansion, though, since $\mathfrak{C}$ and $\mathfrak{S}$ do not coincide. However, it would be straightforward to define the notion of *coupled expansion* as a mutual simulation exhibiting the coupling as required in Definition 2.4.1 such that one of the component simulations is progressing while the other is strict. Another idea that arises immediately is to enhance coupled simulation with the requirement that the two simulations should be eventually progressing. In fact, $(\mathfrak{C}, \mathfrak{S})$ is such an *eventually progressing coupled simulation* as the results in §5 show. Putting all observations together, $(\mathfrak{C}, \mathfrak{S})$ would satisfy the requirements of an eventually progressing coupled expansion being a coupled expansion where the strict component is also progressing. This preorder would in our case express that (1) source terms and target terms are behaviorally the same, (2) source terms are more efficient than target terms since they use fewer τ-steps, and (3) target terms may only diverge if the corresponding source terms do, and vice versa. We think that equivalence notions along this line deserve further investigation.

Axiomatizations of both weak asynchronous bisimulation and asynchronous coupled simulation are not yet known. Alternative formulations of the definitions of asynchronous bisimulation (see [ACS98], also for an axiomatization in the strong case) might prove convenient for finding modal characterizations and also, in general, for establishing bisimulations.

Endomorphic encodings into a language fragment, like the ones which we investigated in this paper, are easier to deal with than encodings between different languages. This fact relies on the use of a common (labeled) transition semantics for the source and target language which allowed us to directly compare terms and translations in a common formal setting. Furthermore, our choice encodings guaranteed that high-level channels are used in exactly the same way in translations. This not always the case, as even for the tuple encoding into monadic π-calculus channels are used differently in source and target. Barbed bisimulation, which was invented to provide a uniform definitional scheme of term equivalence based on reduction semantics [MS92a, HY95], could be especially useful with respect to encodings between different calculi, since it rests on laxer notions of observation. Also, a barbed definition of coupled simulation, based on reduction semantics, might allow to prove results for encodings which are not fully abstract up to weak bisimulation.

Since the writing of the conference version of this paper, new results have been presented concerning more sophisticated choice operators and their encodings. Palamidessi showed in [Pal97] that it is not possible to give a divergence-free encoding of mixed-guarded choice in a fully distributed way, i.e., where parallel composition and restriction do not add centralizing components that could be used to globally manage the interactions on channels. Nestmann showed in [Nes97] that it is, however, possible to encode separate choice (either purely input-guarded or purely output-guarded) up to Palamidessi's criteria. Furthermore, the latter contains a discussion on how to relax the criteria for full distribution to get also 'good' encodings for mixed choice along the lines of [Kna93]. Full abstraction results are not yet known for these more recent encodings.

Acknowledgments

We are indebted to David N. Turner for the original asynchronous choice encodings in Pict, on which our encoding $\mathcal{C}[\![\ ]\!]$ is based. Ole Jensen helped us with clarifying intuitions on the nature of coupled simulation. Kohei Honda helped in improving the paper by providing detailed comments on a draft version. Cédric Fournet suggested the proof technique for Lemma 4.3.4. Robin Milner, Davide Sangiorgi, Peter Sewell, and the rest of the Edinburgh/Cambridge *Pi Club* and the *Dienstagsclub* at Erlangen joined us in many productive discussions.

A Some Proofs

A.1 $\mathcal{F}$ is a strong bisimulation (Proposition 5.3.4)

We show the proof for strong synchronous bisimulation, which implies the asynchronous case; we simply omit the second clause of Definition 2.3.1 and use the first clause also for input transitions. The proof is by structural induction on $A \in \mathbb{A}$ and transition induction on $A \;\rightarrow A'$. By the simplification discussed in Section 5.1, it suffices to regard the case

$$A = \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_B$$

where, according to the rules in Table 5, there are three subcases.

The following proofs are in each case to be read if-and-only-if since the enabling side-conditions for the respective transitions of A and its translation $\mathcal{F}[\![\, A \,]\!]$ coincide in each case. Let $\sigma_j := \{{}^{\mathrm{v}(j)}\!/\!_x\}$ for all $j \in V$.

case ***READ*** In the following J-indexed annotated choice, let $k \in J\backslash(V\cup B)$ and let z be transmittable on y_k and $\mathrm{v}' = \mathrm{v} + (k \mapsto z)$. Let b be defined as t if $B = \emptyset$, and as f otherwise. Then, we have

$$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_B \xrightarrow{y_k z} \big(\sum_{j\in J} R_j\big)^{\mathrm{v}'}_B \qquad \text{and}$$

$$\begin{array}{rlll}
\multicolumn{4}{l}{\mathcal{F}\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_B \,\Big]\!\!\Big]} \\
= & (\boldsymbol{\nu} l)\Big(\;\bar{l}\mathsf{b} \;\mid & \prod_{j\in J\backslash(V\cup B)} & \mathit{Read}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle \\
 & \qquad\quad\;\mid & \prod_{j\in V} & \mathit{Test}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle\sigma_j \;\Big) \\
\xrightarrow{y_k z} & (\boldsymbol{\nu} l)\Big(\;\bar{l}\mathsf{b} \;\mid & \prod_{\boxed{k\neq} j\in J\backslash(V\cup B)} & \mathit{Read}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle \\
 & \qquad\quad\;\mid & & \boxed{\mathit{Test}_l\langle\, \mathcal{F}[\![\, R_k \,]\!] \,\rangle\{{}^{z}\!/\!_x\}} \\
 & \qquad\quad\;\mid & \prod_{j\in V} & \mathit{Test}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle\sigma_j \;\Big) \\
\equiv & (\boldsymbol{\nu} l)\Big(\;\bar{l}\mathsf{b} \;\mid & \prod_{j\in J\backslash(V'\cup B)} & \mathit{Read}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle \\
 & \qquad\quad\;\mid & \prod_{j\in V'} & \mathit{Test}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle\sigma'_j \;\Big) \\
= & \multicolumn{3}{l}{\mathcal{F}\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}'}_B \,\Big]\!\!\Big]}
\end{array}$$

where σ is extended to σ' according to v', yields the proof.

case ***ABORT*** In every context, for $k \in V$, via rule *ABORT*, we have

$$\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B\neq\emptyset} \xrightarrow{\tau} \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{B+k} \mid \overline{y_k}\,\mathrm{v}_k \qquad \text{and}$$

$$
\begin{aligned}
&\mathcal{F}\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B} \,\Big]\!\!\Big] \\
&= \quad (\boldsymbol{\nu} l)\Big(\quad \bar{l}\mathsf{f} \;\Big|\; \prod_{j\in J\setminus(V\cup B)} \mathit{Read}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle \\
&\qquad\qquad\qquad\qquad \Big|\; \prod_{j\in V} \mathit{Test}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle\sigma_j \;\Big) \\
&\xrightarrow{\tau} \quad (\boldsymbol{\nu} l)\Big(\qquad\qquad \prod_{j\in J\setminus(V\cup B)} \mathit{Read}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle \\
&\qquad\qquad\qquad\qquad \Big|\; \prod_{\boxed{k\neq} j\in V} \mathit{Test}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle\sigma_j \\
&\qquad\qquad\qquad\qquad \Big|\; \boxed{\mathit{Abort}_l\langle \mathcal{F}[\![\, R_k \,]\!]\rangle\sigma_k} \;\Big) \\
&\equiv \quad (\boldsymbol{\nu} l)\Big(\quad \bar{l}\mathsf{f} \;\Big|\; \prod_{j\in J\setminus(V\cup B)} \mathit{Read}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle \\
&\qquad\qquad\qquad\qquad \Big|\; \prod_{k\neq j\in V} \mathit{Test}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle\sigma_j \\
&\qquad\qquad\qquad\qquad \Big|\; \overline{y_k}\,\mathrm{v}_k \;\Big) \\
&\equiv \quad \mathcal{F}\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B} \;\Big|\; \overline{y_k}\,\mathrm{v}_k \,\Big]\!\!\Big]
\end{aligned}
$$

which holds since $J\setminus(V-k)\setminus(B+k) = J\setminus(V\cup B)$ and $\overline{y_k}\,\mathrm{v}_k = \overline{y_k}x\sigma_k$.

case *COMMIT* In every context, for $k \in V$, via rule *COMMIT*, we have

$$
\big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\emptyset} \xrightarrow{\tau} \big(\sum_{j\in J} R_j\big)^{\mathrm{v}-k}_{\{k\}} \mid P_k\sigma_k \qquad \text{and}
$$

$$
\begin{aligned}
&\mathcal{F}\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{\emptyset} \,\Big]\!\!\Big] \\
&= \quad (\boldsymbol{\nu} l)\Big(\quad \bar{l}\mathsf{t} \;\Big|\; \prod_{j\in J\setminus(V)} \mathit{Read}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle \\
&\qquad\qquad\qquad\qquad \Big|\; \prod_{j\in V} \mathit{Test}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle\sigma_j \;\Big) \\
&\xrightarrow{\tau} \quad (\boldsymbol{\nu} l)\Big(\qquad\qquad \prod_{j\in J\setminus(V\cup B)} \mathit{Read}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle \\
&\qquad\qquad\qquad\qquad \Big|\; \prod_{\boxed{k\neq} j\in V} \mathit{Test}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle\sigma_j \\
&\qquad\qquad\qquad\qquad \Big|\; \boxed{\mathit{Commit}_l\langle \mathcal{F}[\![\, R_k \,]\!]\rangle\sigma_k} \;\Big) \\
&\equiv \quad (\boldsymbol{\nu} l)\Big(\quad \bar{l}\mathsf{t} \;\Big|\; \prod_{j\in J\setminus(V\cup B)} \mathit{Read}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle \\
&\qquad\qquad\qquad\qquad \Big|\; \prod_{k\neq j\in V} \mathit{Test}_l\langle \mathcal{F}[\![\, R_j \,]\!]\rangle\sigma_j \\
&\qquad\qquad\qquad\qquad \Big|\; \mathcal{F}[\![\, P_k \,]\!]\sigma_k \;\Big) \\
&\equiv \quad \mathcal{F}\Big[\!\!\Big[\, \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B} \;\Big|\; P_k\sigma_k \,\Big]\!\!\Big]
\end{aligned}
$$

which holds because of $J\backslash(V{-}k)\backslash(B{+}k) = J\backslash(V{\cup}B)$ and since substitutions satisfy $\mathcal{F}[\![\, R_k\sigma_k \,]\!] = \mathcal{F}[\![\, R_k \,]\!]\sigma_k$. $\square$

A.2 Operational correspondence for the $\mathcal{B}$-encoding (Proposition 5.5.1)

Proof: We know by $T = \mathcal{F}[\![\, A \,]\!]$ for some $A \in \mathbb{A}$ that all test-subterms of $\mathcal{F}[\![\, A \,]\!]$ are generated as part of some branch in the flattening of an annotated choice. Because of the simplification discussed in Section 5.1, we regard annotated choice terms in isolation.

$$A = \big(\sum_{j\in J} R_j\big)^{\mathrm{v}}_{B}$$

Then, the possible occurrences of test-subterms are those given by

$$\begin{aligned} T := \mathcal{F}[\![\, A \,]\!] \;=\; (l)\Big(\; \overline{l}\mathsf{b} \;\Big|\; & \prod_{j\in J\backslash(V\cup B)} \mathit{Read}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle \\ \Big|\; & \prod_{j\in V} \mathit{Test}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle\{^{\mathrm{v}(j)}\!/\!_x\} \;\Big) \end{aligned}$$

so the $\mathcal{B}$-encoding will only act nontrivially on them.

The proof is by transition induction, where it suffices to regard the cases where transitions involve test-expressions and their translations since other constructs are mapped homomorphically identical by the $\mathcal{B}$-encoding.

1. Immediate since the above test-expressions are always restricted on their channel l such that they can never show any visible behavior.
2. Straightforward, since we only have to follow the encoding's idea. Let us emphasize the structure of T by

   $$T = (l)\Big(\; \overline{l}\mathsf{b} \;\big|\; Q \;\big|\; \mathsf{test}\ l\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2 \;\Big)$$

   (where the test-expression is the derivative of the particular branch that causes the $\mathbb{T}$-level transition, and Q represents the remaining branches). If the rules *TRUE*/*FALSE* are not involved in the derivations, then the simulation by some transition of $\mathcal{B}[\![\, T \,]\!]$ is trivial by the identical inference tree. Otherwise, the transition is of the form

   $$T \xrightarrow{\tau} (l)\Big(\; \mathbf{0} \;\big|\; Q \;\big|\; Q'_i \;\Big) =: T'$$

   where Q'_i is either the continuation process ($i = 1$) of the test-ing branch, or the resent message ($i = 2$), in both cases also providing the message $\overline{l}\mathsf{f}$; we omit the explicit syntax.

In the $\mathcal{B}$-translation, the first action of $\mathcal{B}[\![\, T \,]\!]$ that is necessary in order to simulate the above transition of T has to be on the lock-channel l as becomes clear from the translated term by expanding the 2-adic subterms via the zip-lock encoding:

$$
\begin{array}{rcl}
\mathcal{B}[\![\, T \,]\!] & = & (l) \big(\, \mathcal{Z}[\![\, l(t,f).\overline{b} \,]\!] \\
& & \quad \mid \mathcal{B}[\![\, Q \,]\!] \\
& & \quad \mid (t)(f)(\, \mathcal{Z}[\![\, \overline{l}\langle t,f \rangle \,]\!] \mid t.\mathcal{B}[\![\, Q_1 \,]\!] \mid f.\mathcal{B}[\![\, Q_2 \,]\!] \,) \, \big) \\
& = & (l) \big(\, l(u).(v)(\overline{u}v|v(t).(w)(\overline{u}w|w(f).\overline{b})) \\
& & \quad \mid \mathcal{B}[\![\, Q \,]\!] \\
& & \quad \mid (t)(f) \; (\; (u)(\overline{l}u|u(v).(\overline{v}t|u(w).\overline{w}f)) \\
& & \qquad\qquad \mid t.\mathcal{B}[\![\, Q_1 \,]\!] \\
& & \qquad\qquad \mid f.\mathcal{B}[\![\, Q_2 \,]\!] \;) \, \big) \\
& \xrightarrow{\tau} & (l) \; (u) \; \big(\; (v)(\overline{u}v|v(t).(w)(\overline{u}w|w(f).\overline{b})) \\
& & \qquad \mid \mathcal{B}[\![\, Q \,]\!] \\
& & \qquad \mid (t)(f) \; (\; u(v).(\overline{v}t|u(w).\overline{w}f) \\
& & \qquad\qquad\qquad \mid t.\mathcal{B}[\![\, Q_1 \,]\!] \\
& & \qquad\qquad\qquad \mid f.\mathcal{B}[\![\, Q_2 \,]\!] \;) \, \big) \\
& =: & P
\end{array}
$$

where $b = t$, if $\mathsf{b} = \mathsf{t}$, and $b = f$, otherwise. Note the extrusion of the scope of u, which may now be used as a private communication line between the 2-adic sender and receiver.

In the resulting derivative P, the behavior of the $\mathbb{P}$-level subsystem that belongs to the test-expression—i.e., disregarding $\mathcal{B}[\![\, Q \,]\!]$—is completely determined by four subsequent reductions (interactions along u, v, u, and w, where there is always exactly one sender and receiver) for the exchange of the 2-adic message, such that it arrives at

$$
\begin{array}{rcl}
P & \xrightarrow{\tau}{}^{4} & (l)(t)(f) \; (\; (v)(w)\overline{b} \\
& & \qquad\qquad \mid \mathcal{B}[\![\, Q \,]\!] \\
& & \qquad\qquad \mid (u)\mathbf{0} \\
& & \qquad\qquad \mid t.\mathcal{B}[\![\, Q_1 \,]\!] \\
& & \qquad\qquad \mid f.\mathcal{B}[\![\, Q_2 \,]\!] \,) \, \big) \\
& =: & P' \\
& \equiv_\nu & (l)(t)(f) \; (\; \overline{b} \\
& & \qquad\qquad \mid \mathcal{B}[\![\, Q \,]\!] \\
& & \qquad\qquad \mid t.\mathcal{B}[\![\, Q_1 \,]\!] \\
& & \qquad\qquad \mid f.\mathcal{B}[\![\, Q_2 \,]\!]
\end{array}
$$

where we have done a bit of garbage collection that is possible since the names u, v, and w are no longer used. Finally, after another determinate reduction, depending on the actual value of the $\mathbb{T}$-level lock-message, we get

$$P' \equiv \xrightarrow{\tau} \begin{cases} (l)(t)(f)\big(\,\mathbf{0} \mid \mathcal{B}[\![\, Q \,]\!] \mid \mathcal{B}[\![\, Q_1 \,]\!] \mid f.\mathcal{B}[\![\, Q_2 \,]\!]\,\big) & \\ \equiv (l)(\,\mathcal{B}[\![\, Q \,]\!] \mid \mathcal{B}[\![\, Q_1 \,]\!]\,) & \text{if } b = t \\ & \\ (l)(t)(f)\big(\,\mathbf{0} \mid \mathcal{B}[\![\, Q \,]\!] \mid t.\mathcal{B}[\![\, Q_1 \,]\!] \mid \mathcal{B}[\![\, Q_2 \,]\!]\,\big) & \\ \equiv (l)(\,\mathcal{B}[\![\, Q \,]\!] \mid \mathcal{B}[\![\, Q_2 \,]\!]\,) & \text{if } b = f \end{cases}$$

again by garbage-collecting obsolete name restrictions and inaccessible processes.

3. Straightforward, again by distinguishing the possible cases for τ-steps of $\mathcal{B}[\![\, T \,]\!]$: if the rules *TRUE*/*FALSE* are involved in the derivations, then the begun $\mathbb{P}$-level interaction can be simulated at the $\mathbb{T}$-level since the only test-expressions have come from *Test*-states of branches in annotated choices. □

References

[ACS98] R. M. Amadio, I. Castellani and D. Sangiorgi. On Bisimulations for the Asynchronous π-Calculus. *Theoretical Computer Science*, 195(2):291–324, 1998. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 147–162.

[AH92] S. Arun-Kumar and M. Hennessy. An Efficiency Preorder for Processes. *Acta Informatica*, 29:737–760, 1992.

[ALT95] R. M. Amadio, L. Leth and B. Thomsen. From a Concurrent λ-Calculus to the π-Calculus. In H. Reichel, ed, *Proceedings of FCT '95*, volume 965 of *LNCS*, pages 106–115. Springer, 1995. Full version as Technical Report ECRC-95-18.

[Ama93] R. M. Amadio. On the reduction of CHOCS bisimulation to π-calculus bisimulation. In E. Best, ed, *Proceedings of CONCUR '93*, volume 715 of *LNCS*, pages 112–126. Springer, 1993. Extended version as Rapport de Recherche, INRIA-Lorraine, 1993.

[Ama94] R. M. Amadio. Translating Core Facile. Technical Report ECRC-94-3, European Computer-Industry Research Centre, München, Feb. 1994.

[BG95] N. Busi and R. Gorrieri. Distributed Conflicts in Communicating Systems. In P. Ciancarini, O. Nierstrasz and A. Yonezawa, eds, *Object-Based Models and Languages for Concurrent Systems*, volume 924 of *LNCS*, pages 49–65. Springer, 1995.

[Bou92] G. Boudol. Asynchrony and the π-calculus (Note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[But94] K.-H. Buth. Simulation of SOS Definitions with Term Rewriting Systems. In D. Sannella, ed, *Proceedings of ESOP '94*, volume 788 of *LNCS*, pages 150–164. Springer, 1994.

[FG96] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join-Calculus. In J. G. Steele, ed, *Proceedings of POPL '96*, pages 372–385. ACM, Jan. 1996.

[Gam91] A. Gammelgaard. Constructing Simulations Chunk by Chunk. Internal Report DAIMI IR-106, Computer Science Department, Aarhus University, Dec. 1991. Part of the authors' PhD Thesis *Simulation Techniques*, available as Report DAIMI PB-379.

[Gla93] R. Glabbeek. The Linear Time − Branching Time Spectrum II: The semantics of sequential systems with silent moves (Extended Abstract). In E. Best, ed, *Proceedings of CONCUR '93*, volume 715 of *LNCS*, pages 66–81. Springer, 1993.

[Gla94] D. Gladstein. *Compiler Correctness for Concurrent Languages*. PhD thesis, Northeastern University, Dec. 1994.

[Hon92] K. Honda. Two Bisimilarities in ν-Calculus. CS report 92-002, Keio University, 1992. Revised on March 31, 1993.

[HT91] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communication. In P. America, ed, *Proceedings of ECOOP '91*, volume 512 of *LNCS*, pages 133–147. Springer, July 1991.

[HT92] K. Honda and M. Tokoro. On Asynchronous Communication Semantics. In M. Tokoro, O. Nierstrasz and P. Wegner, eds, *Object-Based Concurrent Computing 1991*, volume 612 of *LNCS*, pages 21–51. Springer, 1992.

[HY94a] K. Honda and N. Yoshida. Combinatory Representation of Mobile Processes. In *Proceedings of POPL '94*, pages 348–360. ACM, January 1994.

[HY94b] K. Honda and N. Yoshida. Replication in Concurrent Combinators. In M. Hagiya and J. C. Mitchell, eds, *Proceedings of TACS '94*, volume 789 of *LNCS*, pages 786–805. Springer, 1994.

[HY95] K. Honda and N. Yoshida. On Reduction-Based Process Semantics. *Theoretical Computer Science*, 152(2):437–486, 1995. An extract appeared in *Proceedings of FSTTCS '93*, LNCS 761.

[Jon93] C. Jones. A Pi-Calculus Semantics for an Object-Based Design Notation. In E. Best, ed, *Proceedings of CONCUR '93*, volume 715 of *LNCS*, pages 158–172. Springer, 1993.

[Kna93] F. Knabe. A Distributed Protocol for Channel-Based Communication with Choice. *Computers and Artificial Intelligence*, 12(5):475–490, 1993.

[Lav93] C. Lavatelli. Non-deterministic Lazy λ-calculus vs. π-calculus. Technical Report LIENS-93-15, Ecole Normale Supérieure, Paris, Sept. 1993.

[Li83] W. Li. *An Operational Approach to Semantics and Translation for Concurrent Programming Languages.* PhD thesis, LFCS, University of Edinburgh, Jan. 1983. Report CST-20-83.

[Li94] B. Z. Li. A π-Calculus Specification of Prolog. In D. Sannella, ed, *Proceedings of ESOP '94*, volume 788 of *LNCS*, pages 379–393. Springer, 1994.

[LT95] L. Leth and B. Thomsen. Some Facile Chemistry. *Formal Aspects of Computing*, 7(3):314–328, 1995. A Previous Version appeared as ECRC-Report ECRC-92-14.

[Mil87] M. Millington. *Theories of Translation Correctness for Concurrent Programming Languages.* PhD thesis, LFCS, University of Edinburgh, Aug. 1987. Report CST-46-87.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mil92] R. Milner. Functions as Processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992. Previous version as Rapport de Recherche 1154, INRIA Sophia-Antipolis, 1990, and in *Proceedings of ICALP '91*, LNCS 443.

[Mil93] R. Milner. The Polyadic π-Calculus: A Tutorial. In F. L. Bauer, W. Brauer and H. Schwichtenberg, eds, *Logic and Algebra of Specification*, volume 94 of *Series F: Computer and System Sciences.* NATO Advanced Study Institute, Springer, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

[Mit86] K. Mitchell. *Implementations of Process Synchronisation and their Analysis.* PhD thesis, LFCS, University of Edinburgh, July 1986.

[MP95] U. Montanari and M. Pistore. Concurrent Semantics for the π-calculus. In S. Brookes, M. Main, A. Melton and M. Mislove, eds, *Proceedings of MFPS '95*, volume 1 of *ENTCS.* Elsevier Science Publishers, 1995.

[MPW92] R. Milner, J. Parrow and D. Walker. A Calculus of Mobile Processes, Part I/II. *Information and Computation*, 100:1–77, Sept. 1992.

[MS92a] R. Milner and D. Sangiorgi. Barbed Bisimulation. In W. Kuich, ed, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[MS92b] U. Montanari and V. Sassone. Dynamic Congruence vs. Progressing Bisimulation for CCS. *Fundamenta Informaticae*, XVI(2):171–199, 1992.

[Nes96] U. Nestmann. *On Determinacy and Nondeterminacy in Concurrent Programming.* PhD thesis, Technische Fakultät, Universität Erlangen, November 1996. Arbeitsbericht IMMD-29(14).

[Nes97] U. Nestmann. What Is a 'Good' Encoding of Guarded Choice? In C. Palamidessi and J. Parrow, eds, *Proceedings of EXPRESS '97*, volume 7 of *ENTCS*. Elsevier Science Publishers, 1997. Latest full version as report BRICS-RS-99-43, Universities of Aalborg and Århus, Denmark, 1999. To appear in *Journal of Information and Computation.*

[Nie96] J. Niehren. Functional Computation as Concurrent Computation. In J. G. Steele, ed, *Proceedings of POPL '96*, pages 333–343. ACM, Jan. 1996. Full version as Research Report RR-95-14, DFKI Saarbrücken.

[Ode95a] M. Odersky. Applying π: Towards a Basis for Concurrent Imperative Programming. In U. S. Reddy, ed, *Proceedings of SIPL '95*, pages 95–108. ACM/SIGPLAN, University of Illinois at Urbana-Champaign, January 22 1995. Available as Technical Report UIUCDCS-R-95-1900.

[Ode95b] M. Odersky. Polarized Name Passing. In P. S. Thiagarajan, ed, *Proceedings of FSTTCS '95*, volume 1026 of *LNCS*, pages 324–337. Springer, 1995.

[OR94] V. Oostrom and F. Raamsdonk. Comparing Combinatory Reduction Systems and Higher-Order Rewrite Systems. In J. Heering, K. Meinke, B. Möller and T. Nipkow, eds, *Proceedings of HOA '93*, volume 816 of *LNCS*, pages 276–304. Springer, 1994.

[Pal97] C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculus. In *Proceedings of POPL '97*, pages 256–265. ACM, Jan. 1997.

[Par99] J. Parrow. Trios in Concert. In G. Plotkin, C. Stirling and M. Tofte, eds, *Proof, Language and Interaction: Essays in Honour of Robin Milner.* MIT Press, 1999. To appear.

[Pri78] L. Priese. On the Concept of Simulation in Asynchronous, Concurrent Systems. In *Progress in Cybernetics and Systems Research*, volume VII, pages 85–92. Hemisphere Dubl. Corp., 1978. Proceedings of EMCSR (1978, Linz, Österreich).

[PS92] J. Parrow and P. Sjödin. Multiway Synchronization Verified with Coupled Simulation. In R. Cleaveland, ed, *Proceedings of CONCUR '92*, volume 630 of *LNCS*, pages 518–533. Springer, 1992.

[PS94] J. Parrow and P. Sjödin. The Complete Axiomatization of cs-Congruence. In P. Enjalbert, E. W. Mayr and K. W. Wagner, eds, *Proceedings of STACS '94*, volume 775 of *LNCS*, pages 557–568. Springer, 1994.

[PT95] B. C. Pierce and D. N. Turner. Concurrent Objects in a Process Calculus. In T. Ito and A. Yonezawa, eds, *Proceedings of TPPP '94*, volume 907 of *LNCS*, pages 187–215. Springer, 1995.

[PT99] B. C. Pierce and D. N. Turner. Pict: A Programming Language Based on the Pi-Calculus. In G. Plotkin, C. Stirling and M. Tofte, eds, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1999. To appear.

[Ros93] B. Ross. A π-calculus semantics of logical variables and unification. In S. Purushothaman and A. Zwarico, eds, *Proceedings of First North American Process Algebra Workshop, Stony Brook, 1992*, Workshops in Computing, pages 216–230. Springer, 1993.

[San93] D. Sangiorgi. From π-calculus to Higher-Order π-calculus — and back. In M.-C. Gaudel and J.-P. Jouannaud, eds, *Proceedings of TAPSOFT '93*, volume 668 of *LNCS*, pages 151–166. Springer, 1993.

[San94a] D. Sangiorgi. An investigation into functions as processes. In S. Brookes, M. Main, A. Melton and D. Schmidt, eds, *Proceedings of MFPS '93*, volume 802 of *LNCS*, pages 143–159. Springer, 1994.

[San94b] D. Sangiorgi. The Lazy Lambda Calculus in a Concurrency Scenario. *Information and Computation*, 111(1):120–131, 1994. An extended abstract appeared in *Proceedings of LICS '92*.

[San95a] D. Sangiorgi. Lazy functions and mobile processes. Rapport de Recherche RR-2515, INRIA Sophia-Antipolis, 1995.

[San95b] D. Sangiorgi. On the Bisimulation Proof Method. In J. Wiedemann and P. Hájek, eds, *Proceedings of MFCS '95*, volume 969 of *LNCS*, pages 479–488. Springer, 1995.

[San96] D. Sangiorgi. π-Calculus, Internal Mobility and Agent-Passing Calculi. *Theoretical Computer Science*, 167(1,2):235–274, 1996. Also as Rapport de Recherche RR-2539, INRIA Sophia-Antipolis, 1995. Extracts of parts of the material contained in this paper can be found in *Proceedings of TAPSOFT '95* and *ICALP '95*.

[Smo94] G. Smolka. A Foundation for Higher-Order Concurrent Constraint Programming. In J.-P. Jouannaud, ed, *Proceedings 1st International Conference of Constraints in Computational Logics*, volume 845 of *LNCS*, pages 50–72. Springer, 1994. Available as Research Report RR-94-16 from DFKI Kaiserslautern.

[Tho93] B. Thomsen. Plain CHOCS. A Second Generation Calculus for Higher Order Processes. *Acta Informatica*, 30(1):1–59, 1993.

[Tho95] B. Thomsen. A Theory of Higher Order Communicating Systems. *Information and Computation*, 116(1):38–57, 1995.

[Vaa90] F. Vaandrager. Process Algebra Semantics of POOL. In J. Baeten, ed, *Application of Process Algebra*, pages 173–236. Cambridge University Press, 1990. Earlier version: CWI-Report CS-R8629.

[VP96] B. Victor and J. Parrow. Constraints as Processes. In U. Montanari and V. Sassone, eds, *Proceedings of CONCUR '96*, volume 1119 of *LNCS*, pages 389–405. Springer, 1996.

[Wal90] D. Walker. Bisimulation and Divergence. *Information and Computation*, 85(2):202–241, 1990. An extended abstract appeared in the *Proceedings of LICS' 88*.

[Wal91a] D. Walker. π-calculus Semantics of Object-Oriented Programming Languages. In T. Ito and A. Meyer, eds, *Proceedings of TACS '91*, volume 526 of *LNCS*, pages 532–547. Springer, 1991. Available as Report ECS-LFCS-90-122, University of Edinburgh.

[Wal91b] D. Walker. Some Results on the π-Calculus. In A. Yonezawa and T. Ito, eds, *Concurrency: Theory, Languages, and Architecture*, volume 491 of *LNCS*, pages 21–35. Springer, 1991.

[Wal93] D. Walker. Process Calculus and Parallel Object-Oriented Programming Languages. In P. Tvrdik, T. Casavant and F. Plasil, eds, *Parallel Computers: Theory and Practice*, pages 369–390. Computer Society Press, 1993. Available as Research Report CS-RR-242, Department of Computer Science, University of Warwick.

[Wal94] D. Walker. Algebraic Proofs of Properties of Objects. In D. Sannella, ed, *Proceedings of ESOP '94*, volume 788 of *LNCS*, pages 501–516. Springer, 1994.

[Wal95] D. Walker. Objects in the π-calculus. *Information and Computation*, 116(2):253–271, 1995.

Recent BRICS Report Series Publications

RS-99-42 **Uwe Nestmann and Benjamin C. Pierce.** ***Decoding Choice Encodings*****. December 1999. ii+62 pp. To appear in** ***Journal of Information and Computation*****. An extended abstract appeared in Montanari and Sassone, editors,** ***Concurrency Theory: 7th International Conference*****, CONCUR '96 Proceedings, LNCS 1119, 1996, pages 179–194.**

RS-99-41 **Nicky O. Bodentien, Jacob Vestergaard, Jakob Friis, Kåre J. Kristoffersen, and Kim G. Larsen.** ***Verification of State/Event Systems by Quotienting*****. December 1999. 17 pp. Presented at** ***Nordic Workshop in Programming Theory*****, Uppsala, Sweden, October 6–8, 1999.**

RS-99-40 **Bernd Grobauer and Zhe Yang.** ***The Second Futamura Projection for Type-Directed Partial Evaluation*****. November 1999. Extended version of an article to appear in Lawall, editor,** ***ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation*****, PEPM '00 Proceedings, 2000.**

RS-99-39 **Romeo Rizzi.** ***On the Steiner Tree*** $\frac{3}{2}$***-Approximation for Quasi-Bipartite Graphs*****. November 1999. 6 pp.**

RS-99-38 **Romeo Rizzi.** ***Linear Time Recognition of*** P_4***-Indifferent Graphs*****. November 1999. 11 pp.**

RS-99-37 **Tibor Jordán.** ***Constrained Edge-Splitting Problems*****. November 1999. 23 pp. A preliminary version with the title** ***Edge-Splitting Problems with Demands*** **appeared in Cornujols, Burkard and Wöginger, editors,** ***Integer Programming and Combinatorial Optimization: 7th International Conference*****, IPCO '99 Proceedings, LNCS 1610, 1999, pages 273–288.**

RS-99-36 **Gian Luca Cattani and Glynn Winskel.** ***Presheaf Models for*** **CCS-*****like Languages*****. November 1999. ii+46 pp.**

RS-99-35 **Tibor Jordán and Zoltán Szigeti.** ***Detachments Preserving Local Edge-Connectivity of Graphs*****. November 1999. 16 pp.**

RS-99-34 **Flemming Friche Rodler.** ***Wavelet Based 3D Compression for Very Large Volume Data Supporting Fast Random Access*****. October 1999. 36 pp.**