Decoding Choice Encodings*

Uwe Nestmann[†] Benjamin C. Pierce[‡]

April 1996

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 require bisimilarity of internal branching decisions. The correctness proofs for the two choice encodings exploit the properties of *decodings* from translations to source terms.

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 expressivity 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, PT96], 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.

*University of Cambridge: Technical Report 392; Friedrich-Alexander-Universität Erlangen-Nürnberg: Interner Bericht IMMD VII-01/96.

†`Uwe.Nestmann@informatik.uni-erlangen.de`, Friedrich-Alexander-Universität Erlangen-Nürnberg, Informatik VII, Martensstraße 3, D-91058 Erlangen. Supported by *Deutsche Forschungsgemeinschaft*, Sonderforschungsbereich 182, project C2, and by *Deutscher Akademischer Austauschdienst* within the ARC-program.

‡`Benjamin.Pierce@cl.cam.ac.uk`, University of Cambridge, Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK. Supported by the British Science and Engineering Research Council.

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 labelled semantics [HT91, HT92, Hon92]; the mini π-calculus used a chemical semantics [Bou92]; it has also been investigated using reduction semantics [HY93], 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 labelled semantics, and studied with bisimulation from an asynchronous observer's viewpoint [San95d, ACS96].

We use standard notation for the restriction $(\boldsymbol{\nu} x)\, P$ of name x to process P, for parallel composition $P_1|P_2$, input $y(x).P$ of a name x from channel y for use in P, and output $\overline{y}\langle z\rangle$ of name z on channel y. Furthermore, $\prod$ and $\sum$ denote indexed parallel composition and input-guarded choice, respectively. For convenience, we use the conditional form if ... then ... else ... which performs a case analysis driven by special names $\boldsymbol{t}$ and $\boldsymbol{f}$.

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

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

runs a mutual exclusion protocol, installing a local lock on the parallel composition of its branches. The branches

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

concurrently try to acquire the lock after reading messages from the environment. Only the first branch managing to interrogate the lock will proceed with its continuation and thereby commit the choice — every other branch will then be forced to resend its message and abort its continuation. 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 intuitively seems to be correct.

However, even for an asynchronous observer, it turns out that source terms and their $\mathcal{C}$-translations are not weakly bisimilar. The reason is that $\mathcal{C}$-translations of choice carry out commitments only gradually, resulting in intermediate states which do not correspond any source term. In order to deal with partially committed states, we instead characterize the correctness of the encoding as a pair of opposite 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 that allows 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 (Section 2), followed by the divergence-free encoding $\mathcal{C}$ and the divergent encoding $\mathcal{D}$ (Section 3). We review some standard notions of correctness (Section 4) and present an example that invalidates the full abstraction of the $\mathcal{C}$-encoding with respect to weak asynchronous bisimulation (Section 5). 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 $S \approx \mathcal{D}[\![\, S \,]\!]$ holds (Section 6). Finally, we offer some concluding remarks and sketch related and future work (Section 7).

2 Technical preliminaries

Many variants of the π-calculus [MPW89] have appeared in the recent process algebra literature. We use here a version which is close to the core language of Pict [PT95, PT96], but slightly simplified in order to shorten the presentation in this paper. It is an asynchronous, first-order, monadic π-calculus [HT91, Bou92]. Replication is restricted to input processes and evaluated lazily [HY93, MP95]. Furthermore, merely for notational convenience, it is equipped with a conditional form based on special boolean names, reminiscent of the π_v-calculus [Wal94], but not allowing for more complex boolean expressions; the conditional can be seen as a very restricted form of matching. The choice operator is finite and all the branches of a choice must be guarded by input prefixes (in [ACS96], there are also τ-guards).

2.1 Syntax

Let $\mathbf{N}$ be a countable set of *names*. Let the *booleans* $\mathbf{B}$ be $\{\boldsymbol{t}, \boldsymbol{f}\}$ with $\mathbf{B} \cap \mathbf{N} = \emptyset$ and *values* $\mathbf{V}$ be $\mathbf{N} \cup \mathbf{B}$. The set $\mathbb{P}$ of *processes* P is defined by the following grammar

$$\begin{array}{lcl} R & ::= & y(x).P \\ P & ::= & \mathbf{0} \;\big|\; (\boldsymbol{\nu} x)\, P \;\big|\; P|P \;\big|\; \mathsf{if}\ z\ \mathsf{then}\ P\ \mathsf{else}\ P \;\big| \\ & & \overline{y}\langle z\rangle \;\big|\; R \;\big|\; !R \;\big|\; \sum_{j\in J} R_j \end{array}$$

where $y, z \in \mathbf{V}$, $x \in \mathbf{N}$, and J ranges over finite sets of indices. We also use the abbreviation $R_1 + R_2$ to denote binary choice. Operator precedence is, in decreasing order of binding strength: (1) prefixing, restriction, replication, (2) substitution, (3) parallel composition, and (4) choice. A term is *guarded* when it occurs as a subterm of an input prefix.

The operational semantics of restriction, parallel composition, conditional, input and output is standard. The form $!R$ denotes the replication operator restricted to input-prefixes. In $\overline{y}\langle z\rangle$ and $y(x)$, the name y is called the *subject*, whereas x, z are called *objects*. We refer to outputs as *messages* and to input, replicated input, and choice as *receptors*.

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 $(\boldsymbol{\nu} x)$ or an input prefix $y(x)$ as a subterm. A name x is *free* in P if it occurs outside the scope of an x-binding operator. We write $\mathit{bn}(P)$ and $\mathit{fn}(P)$ for the sets of P's bound and free names; $\mathit{n}(P)$ 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 if necessary to avoid capture. Use of the boolean constants $\boldsymbol{t}$ and $\boldsymbol{f}$ as binding variables is forbidden.

2.2 Operational semantics

Let $y \in \mathbf{N}$ and $z \in \mathbf{V}$ be an arbitrary name and value. The set $\mathbf{L}$ of *labels* μ is generated by

$$\mu \;::=\; \overline{y}\langle \boldsymbol{\nu} z\rangle \;\;\Big|\;\; \overline{y}\langle z\rangle \;\;\Big|\;\; y\langle z\rangle \;\;\Big|\;\; \tau$$

representing the bound and free output, early input, and internal action. The functions bn and fn yield the bound names (those marked by $\boldsymbol{\nu}$ in bound outputs) and free names (all others) of a label. Let $\mathit{n}(\mu) = \mathit{bn}(\mu) \cup \mathit{fn}(\mu)$ denote the set of names occurring in label μ.

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 rules *INP* and *COM/CLOSE* since it allows us to define bisimulation without explicit name-instantiation and since it allows for a more intuitive modelling in Section 6.2, 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.

As usual, weak arrows $\Rightarrow$ denote the reflexive and transitive closure of internal transitions; arrows with hats allow us to specify that two processes are either related by a particular transition or else equal in the case of an internal transition.

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

2.3 Bisimulation

Two process systems are regarded as equivalent when they allow us to observe the same operational behavior at their interface to the environment. Bisimulation defines equivalence as mutual simulation of single computation steps resulting in equivalent system states. According to the standard literature on (ground) bisimulations, e.g. [Mil89, MPW89], a *simulation* is a relation $\mathcal{S}$ on agents such that $(P, Q) \in \mathcal{S}$ implies, for arbitrary label μ:

$$\textit{E-INP:}\quad y(x).P \xrightarrow{y\langle z\rangle} P\{^{z}\!/_{x}\}$$

$$\textit{R-INP:}\quad !\,y(x).P \xrightarrow{y\langle z\rangle} P\{^{z}\!/_{x}\} \mid !\,y(x).P$$

$$\textit{C-INP:}\quad \sum_{j\in J} y_j(x).P_j \xrightarrow{y\langle z\rangle} P_k\{^{z}\!/_{x}\} \qquad \textit{if } y_k = y \textit{ for some } k \in J$$

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

$$\textit{COM}^{*}\textit{:}\quad \frac{P \xrightarrow{\overline{y}\langle z\rangle} P' \qquad Q \xrightarrow{y\langle z\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

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

$$\textit{OPEN:}\quad \frac{P \xrightarrow{\overline{y}\langle x\rangle} P'}{(\boldsymbol{\nu}x)\,P \xrightarrow{\overline{y}\langle\boldsymbol{\nu}x\rangle} P'} \qquad \textit{if } y \neq x$$

$$\textit{CLOSE}^{*}\textit{:}\quad \frac{P \xrightarrow{\overline{y}\langle\boldsymbol{\nu}z\rangle} P' \qquad Q \xrightarrow{y\langle z\rangle} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}z)\,(\,P' \mid Q'\,)} \qquad \textit{if } z \notin \mathsf{fn}(Q)$$

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

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

$$\textit{TRUE:}\quad \frac{P \xrightarrow{\mu} P'}{\mathsf{if}\ \boldsymbol{t}\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\mu} P'}$$

$$\textit{FALSE:}\quad \frac{Q \xrightarrow{\mu} Q'}{\mathsf{if}\ \boldsymbol{f}\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\mu} Q'}$$

* *and the evident symmetric rules*

Table 1: Transition semantics

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

The central idea is that of an external observer that performs experiments with a process P under observation. In an output-experiment, the observer tries to receive messages from the process, which is possible if the process has a matching output transition; in an input-experiment, the observer tries to send messages to the process, which is successful if the process has the matching input transition; finally, the observer may also notice that a process engages in an internal transition, which could be called a reduction-experiment. 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 handle divergence.

Asynchrony

In the calculus with synchronous output, the existence of an input transition $P \xrightarrow{y\langle z\rangle} P'$ precisely models the success of an observer's input-experiment. Since input labels are only generated for P, if there is a matching receptor sitting enabled inside P, the existence of this transition tells that in P' the message offered by the observer has actually been consumed.

The concept of asynchronous messages suggests a different notion of input-experiment. Asynchronous output is performed independently from the availability of a matching receptor. Consequently, an asynchronous observer sending messages into a configuration cannot see directly whether its messages are consumed or not. Hence, the modelling of an input-experiment by existence of input transitions seems to be too strong since, there, the input-derivative P' definitely has consumed the message.

However, an asynchronous observer may see indirectly that its messages have been consumed by noticing messages that eventually come back from the process. Consequently, instead of abandoning input-experiments completely, asynchronous observation captures input-experiments indirectly by performing output-experiments in the context of arbitrary messages. These considerations have also led to the development of *output-only barbed congruence* [Ode95a, FG96] based on reduction semantics [HY93]. There is an analogy between asynchronous observers and internet users that communicate with each other via email: assuming that email is the only way to exchange information, a user can not see that his email has been read unless his communication partner has sent off some response.

Two different formulations of *asynchronous bisimulation* have been proposed.

Honda and Tokoro [HT91, HT92, Hon92] introduced a modified input rule in order to model asynchronous input-experiments explicitly:

$$P \xrightarrow{y\langle z\rangle} P \mid \overline{y}\langle z\rangle$$

This rule allows for any system at any time to accept an arbitrary message without the necessity of containing a matching receptor consuming it. The standard technique of performing input-experiments by checking the existence of input transitions obviously applies with this type of semantics. Note that the modified transition relation is no longer characterizing the computational content of processes, but more describing their observational behavior.

In contrast, Sangiorgi suggested that we keep to the standard labelled semantic rules, but instead incorporate the asynchronous style of input-experiments into the definition of simulation such that inputs of processes have to be simulated only indirectly by observing the processes' output behavior in the context of arbitrary messages [San95d, ACS96].

In this paper, we follow the latter approach.

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

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

Bisimulation in name-passing calculi is not a congruence by itself. The main obstacle is that it is not preserved by name-instantiation in the presence of matching and summation [MPW89]. Nevertheless, if we restrict our attention to the subset $\mathbb{S}$ of processes S where in occurrences of a conditional $\mathsf{if}\ z\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2$, the value z is either a special name or bound in S, then bisimulation in this language is preserved by name-instantiation (see the end of Section 3 for more discussion of $\mathbb{S}$).

Lemma 2.3.2 *$\approx$ is a congruence on $\mathbb{S}$.*

Proof: Weak asynchronous bisimulation $\approx$ has been shown to be preserved by name-instantiation on $\mathbb{P}$ without conditionals [ACS96] (see also [Hon92] for the corresponding result, apart from choice, in the ν-calculus). Since we only allow input-guarded choice (there are no τ-guards) we also have the congruence of $\approx$ for our choice operator. Because of the conditional, $\approx$ is preserved by all $\mathbb{P}$-operators but input prefix, due to the standard example in π-calculi with matching. Here, the problem arises from processes $S = \mathsf{if}\ z\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2$ where z is not a special name. Then S is bisimilar to $\mathbf{0}$, but may behave differently when placed in the context of an input prefix that uses z as the input variable. Hence, by receiving a special name, the conditional may become activated and therefore behave different from $\mathbf{0}$. For $\mathbb{S}$, as defined above, we have excluded the possibility that processes may have subterms like S. $\square$

α-conversion	P	$\equiv$	Q	*if* $P =_\alpha Q$
true	$\mathsf{if}\ \boldsymbol{t}\ \mathsf{then}\ P\ \mathsf{else}\ Q$	$\equiv$	P	
false	$\mathsf{if}\ \boldsymbol{f}\ \mathsf{then}\ P\ \mathsf{else}\ Q$	$\equiv$	Q	
associativity	$P \mid (Q \vert R)$	$\equiv$	$(P \vert Q) \mid R$	
commutativity	$P \mid Q$	$\equiv$	$Q \mid P$	
scope extrusion	$(\boldsymbol{\nu} y)\, P \mid Q$	$\equiv$	$(\boldsymbol{\nu} y)\, (P \vert Q)$	*if* $y \notin \mathsf{fn}(Q)$
scope elimination	$(\boldsymbol{\nu} y)\, Q$	$\equiv$	Q	*if* $y \notin \mathsf{fn}(Q)$

Table 2: Structural laws

Structure

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 reduction semantics [MS92, HY93], a structural congruence relation is adopted *a priori* in order to allow for simplified presentations of the operational rules. In this paper, we use the structural laws ($\equiv$) listed in Table 2 only in order to simplify the presentation of some derivation sequences of transitions.

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

In particular, we always work *up to* α-conversion, i.e. we omit to mention the implicit application of rule *ALPHA* since it is captured by a structural law. Thus, we 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. Thus, its definition pays more attention to internal steps.

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

- faithful, *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 $\mathsf{bn}(\mu) \cap \mathsf{fn}(P \vert Q) = \emptyset$.

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

Definition 2.3.5 (Expansion) *A binary relation $\mathcal{E}$ on processes is an* expansion *if $\mathcal{E}$ is a faithful weak simulation and $\mathcal{E}^{-1}$ is a strict weak 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

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 bisimulation is not sensitive with respect to the divergence of processes. This lack of expressivity 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, 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 [Pri78]) to make precise what this means.

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

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

According to the definition, a simulation $\mathcal{S}$ is 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 faithful simulation is progressing by definition 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 cause progress. This resembles the *chunk-by-chunk* idea of simulation in [Gam91].

Lemma 2.3.8 *Progressing simulations preserve divergence.*

Proof: Let $\mathcal{S}$ be a progressing simulation and $(P, Q) \in \mathcal{S}$. If $P \Uparrow$ then there is $P \stackrel{\tau}{\longrightarrow}^\omega$. Since $\mathcal{S}$ is progressing, there is $k_P \in \mathbb{N}$ such that $P \stackrel{\tau}{\longrightarrow}^{k_P+1} P' \stackrel{\tau}{\longrightarrow}^\omega$ and $Q \stackrel{\tau}{\longrightarrow}^+ Q'$ 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 CCS-processes

$$P \equiv \tau.a + \tau.b + \tau.c \qquad \text{and} \qquad Q \equiv \tau.a + \tau.(\ \tau.b + \tau.c\ ).$$

Whereas in P the choice between a, b, and c is atomic, there is a *gradual commitment* going on in Q. There, in order to choose b, first a has to be pre-empted, then by another internal step the choice is resolved by pre-empting c. P and Q might be seen as equivalent when disregarding the internal choices which are present in Q but not in P; however, they are not weakly bisimilar. At best, we can find two contrary simulations $\mathcal{S}_1$ and $\mathcal{S}_2^{-1}$ with

$$\begin{array}{rcl} \mathcal{S} & \stackrel{\text{def}}{=} & \{\ (P,Q),\ (a,a),\ (b,b),\ (c,c),\ (\mathbf{0},\mathbf{0})\ \} \\ \mathcal{S}_1 & \stackrel{\text{def}}{=} & \mathcal{S} \cup \{\ (b,\ \tau.b+\tau.c),\ (c,\ \tau.b+\tau.c)\ \} \\ \mathcal{S}_2 & \stackrel{\text{def}}{=} & \mathcal{S} \cup \{\ (P,\ \tau.b+\tau.c)\ \} \end{array}$$

which, unfortunately, do not coincide. The distinguishing pairs express the problem with the *partially committed* τ-derivative $\tau.b{+}\tau.c$ of Q that cannot be simulated by any nontrivial τ-derivative of P and itself has lost the ability of simulating P.

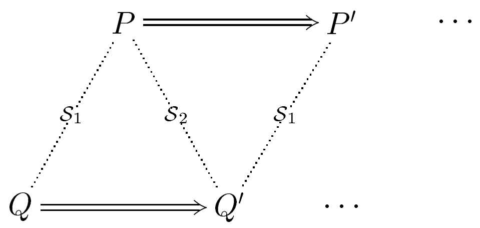
As an appropriate mathematical tool to handle situations as the above CCS-example, Parrow and Sjödin developed the notion of *coupled simulation* [PS92]: two contrary simulations are no longer required to coincide, but only to be coupled in a certain way. Several candidates have been presented for what it means to be coupled. No coupling at all would lead to the notion of trace equivalence. 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]. In this paper, we use a generalization for divergent processes, as suggested in [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*

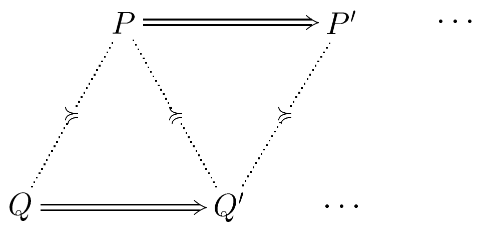
- *if* $(P,Q) \in \mathcal{S}_1$*, then there is some* Q' *such that* $Q \Rightarrow Q'$ *and* $(P,Q') \in \mathcal{S}_2$*;*
- *if* $(P,Q') \in \mathcal{S}_2$*, then there is some* P' *such that* $P \Rightarrow P'$ *and* $(P',Q') \in \mathcal{S}_1$*.*

Two processes P *and* Q *are* 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:



or else by:



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'.

Lemma 2.4.2 $\rightleftarrows$ *is an equivalence.*

Proof: Reflexivity and symmetry are immediate.

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 $\mathcal{S}_{PQR} \stackrel{\text{def}}{=} \mathcal{S}_{PQ}\mathcal{S}_{QR}$ and $\mathcal{S}_{RQP} \stackrel{\text{def}}{=} \mathcal{S}_{QP}\mathcal{S}_{RQ}$. 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 $(P,R'') \in \mathcal{S}_{QP}\mathcal{S}_{RQ} = \mathcal{S}_{RQP}$. □

Proposition 2.4.3 $\rightleftarrows$ *is a congruence on* $\mathbb{S}$.

Proof: Similar to the proposition for $\approx$. □

Fact 2.4.4 $\approx \;\subset\; \rightleftarrows \;\subset\; \preccurlyeq$.

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 relation is denoted by juxtaposition.)

Definition 2.5.1 (Weak 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 $Q \overset{\widehat{\mu}}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{S} \lesssim$.*
- *$(\ \overline{a}\langle z\rangle|P\ ,\ \overline{a}\langle z\rangle|Q\ ) \in \mathcal{S}$ for arbitrary messages $\overline{a}\langle z\rangle$.*

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 $\mathcal{U} \cup \{\ (P', Q') \mid \exists (P_0, Q_0) \in \mathcal{U}.\ \exists (P', Q'') \in \mathcal{U}.\ \exists \mu \in \mathbf{L}.\ (P \xrightarrow{\mu} P' \wedge Q \overset{\widehat{\mu}}{\Longrightarrow} Q'' \lesssim Q')\ \}$ 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 Encoding choice

This section 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 both functions coincide on all constructors but choice, we use a common homomorphic scheme of definition, where $[\![\ ]\!]$ may denote either $\mathcal{C}[\![\ ]\!]$ or $\mathcal{D}[\![\ ]\!]$:

$$\begin{array}{rclrcl}
[\![\,\mathbf{0}\,]\!] & \stackrel{\text{def}}{=} & \mathbf{0} & [\![\,P_1|P_2\,]\!] & \stackrel{\text{def}}{=} & [\![\,P_1\,]\!] \mid [\![\,P_2\,]\!] \\
[\![\,\overline{y}\langle z\rangle\,]\!] & \stackrel{\text{def}}{=} & \overline{y}\langle z\rangle & [\![\,(\boldsymbol{\nu}x)\,P\,]\!] & \stackrel{\text{def}}{=} & (\boldsymbol{\nu}x)\,[\![\,P\,]\!] \\
[\![\,y(x).P\,]\!] & \stackrel{\text{def}}{=} & y(x).[\![\,P\,]\!] & [\![\,!\,P\,]\!] & \stackrel{\text{def}}{=} & !\,[\![\,P\,]\!]
\end{array}$$

$$[\![\,\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2\,]\!] \stackrel{\text{def}}{=} \mathsf{if}\ b\ \mathsf{then}\ [\![\,P_1\,]\!]\ \mathsf{else}\ [\![\,P_2\,]\!]$$

Two slightly different ways of implementing choices will be considered in the following subsections, differing only with respect to the possibility of undoing activities of branches.

3.1 Divergence-free protocol

In the introduction, we presented the following algorithm:

$$\mathcal{C}[\![\,\sum_{j\in J} y_j(x).P_j\,]\!] = (\boldsymbol{\nu}l)\Big(\ \overline{l}\langle \boldsymbol{t}\rangle \ \mid\ \prod_{j\in J} y_j(x).l(b).\big(\ \mathsf{if}\ b\ \mathsf{then}\ \mathcal{C}[\![\,P_j\,]\!]\ \mathsf{else}\ \overline{y_j}\langle x\rangle \ \mid\ \overline{l}\langle \boldsymbol{f}\rangle\ \big)\ \Big)$$

Mutual exclusion is implemented by a boolean message on the fresh local channel l, which plays the role of a lock. Initially it carries $\boldsymbol{t}$, representing the fact that the choice is not yet resolved. Each time the lock is read, it is immediately reinstalled with the value $\boldsymbol{f}$. Thus, at most one branch will ever be chosen. Every branch must interrogate the lock *after* having received a value on its channel. If it is the first to read the lock, then it proceeds; otherwise it resends the message that it has consumed and terminates.

In order to conveniently denote intermediate states in the branches of an encoding, we use the following abbreviations.

$$\begin{array}{rcl}
\mathit{Init}_l\langle\, y(x).P\,\rangle & \stackrel{\text{def}}{=} & y(x).\mathit{Lock}_l\langle\, y(x).P\,\rangle \\
\mathit{Lock}_l\langle\, R\,\rangle & \stackrel{\text{def}}{=} & l(b).\mathit{Exit}_l^b\langle\, R\,\rangle \\
\mathit{Exit}_l^b\langle\, R\,\rangle & \stackrel{\text{def}}{=} & \mathsf{if}\ b\ \mathsf{then}\ \mathit{Commit}_l\langle\, R\,\rangle\ \mathsf{else}\ \mathit{Abort}_l\langle\, R\,\rangle \\
\mathit{Commit}_l\langle\, y(x).P\,\rangle & \stackrel{\text{def}}{=} & \overline{l}\langle \boldsymbol{f}\rangle \mid P \\
\mathit{Abort}_l\langle\, y(x).P\,\rangle & \stackrel{\text{def}}{=} & \overline{l}\langle \boldsymbol{f}\rangle \mid \overline{y}\langle x\rangle
\end{array}$$

A choice over input prefixes R_j is translated by

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

into the composition of its branches and the lock in initial state. As the reader may easily verify, this definition coincides with the former one-line definition.

3.2 Protocol with undo-loops

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

Let *internal choice* be encoded by

$$P \oplus Q \ \stackrel{\text{def}}{=}\ (\boldsymbol{\nu} i, s)\ (\ \overline{i}\langle s\rangle \ \mid\ i(s).P \ \mid\ i(s).Q\ ) \qquad \text{with } i, s \text{ fresh}$$

The encoding $\mathcal{D}[\![\ ]\!]$ is then defined by modifying the $\mathcal{C}$-encoding as follows:

$$\begin{aligned}
\mathit{Init}_l\langle\, y(x).P\,\rangle &\ \stackrel{\text{def}}{=}\ y(x).\mathit{Lock}_l\langle\, y(x).P\,\rangle \\
\mathit{Lock}_l\langle\, R\,\rangle &\ \stackrel{\text{def}}{=}\ l(b).\mathit{Exit}^b_l\langle\, R\,\rangle \\
\mathit{Exit}^b_l\langle\, R\,\rangle &\ \stackrel{\text{def}}{=}\ \mathsf{if}\ b\ \mathsf{then}\ \mathit{Commit}_l\langle\, R\,\rangle \oplus \mathit{Undo}_l\langle\, R\,\rangle\ \mathsf{else}\ \mathit{Abort}_l\langle\, R\,\rangle \\
\mathit{Commit}_l\langle\, y(x).P\,\rangle &\ \stackrel{\text{def}}{=}\ \overline{l}\langle \boldsymbol{f}\rangle \mid P \\
\mathit{Abort}_l\langle\, y(x).P\,\rangle &\ \stackrel{\text{def}}{=}\ \overline{l}\langle \boldsymbol{f}\rangle \mid \overline{y}\langle x\rangle \\
\mathit{Undo}_l\langle\, y(x).P\,\rangle &\ \stackrel{\text{def}}{=}\ \overline{l}\langle \boldsymbol{t}\rangle \mid \overline{y}\langle x\rangle
\end{aligned}$$

Note the difference in the lock's value which is present in the cases of *Commit*/*Abort* and *Undo*. It is crucial to reinstall $\boldsymbol{t}$ in the case that a successfully activated branch undoes its activity. In order to have a fresh copy of the branch in initial state available after having undone an activity, we use replication:

$$\mathcal{D}[\![\sum_{j\in J} R_j]\!] \ \stackrel{\text{def}}{=}\ (\boldsymbol{\nu} l)\left(\ \overline{l}\langle \boldsymbol{t}\rangle \ \mid\ \prod_{j\in J} !\, \mathit{Init}_l\langle\, \mathcal{D}[\![R_j]\!]\,\rangle\ \right) \qquad \text{where } l \text{ is fresh}$$

In their $\mathcal{D}$-translation, convergent branches of a choice term possibly engage in internal loops. The intention of the encoding is to use those loops to restart a possibly committing branch from an initial state. Our encoding also allows loops for non-chosen branches, which could be avoided if, instead of replication of branches, we used recursive *Init*-constants that were only accessible when the choice is not yet resolved. However, we would only gain a bit of efficiency in this way, and the present encoding is simpler since it involves fewer internal states. Note that the $\mathcal{C}$-encoding does not add divergence to the behavior of source terms, as can be observed by inspection of the *Lock*- and *Exit*-abbreviations (we prove it formally in Subsection 6.7). In Section 5, we shall see that the $\mathcal{D}$-encoding is interesting despite its divergence.

3.3 Primitive booleans?

We have chosen to use a language with primitive booleans and conditional form in order to formulate the encoding as crisply as possible. Note that it is not essential to use those primitive forms, as the reformulation of the $\mathcal{C}$-encoding without them, but now in the setting of a polyadic π-calculus, in Appendix A.1 shows. The price that we pay for using $\mathsf{if} \ldots \mathsf{then} \ldots \mathsf{else}$ is that weak bisimulation is not a congruence for the whole language $\mathbb{P}$, but only for the subset $\mathbb{S}$. We do not care about this restriction since we are only interested in using conditionals as convenient programming form within the encoding functions; we are satisfied with results about source terms in which no booleans occur at all. Those terms, and also their translations, live within the subset $\mathbb{S}$.

4 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 characterizing the class of encodings which is represented by the two choice encodings of the previous section.

Intuitively, we require that every source term S and its translation $[\![S]\!]$ should be **semantically equivalent** and interchangeable in any term context, i.e. congruent,

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

where $\asymp$ denotes some notion of equivalence. The stronger the 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 which 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:

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

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.[1] Then, operational correspondence is characterized by two complementary propositions, which we 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, which allow administrative (or book-keeping) steps *preceding* a committing step, unfortunately, do not satisfy $\mathfrak{I}$. Sometimes, as in [Ama94, ALT95], they are well behaved in that pre-administrative steps 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, such that $\mathfrak{I}$ holds. Yet, there is another property taking pre-administrative steps into account:

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

[1] Though we use unlabelled transitions in this discussion, the correctness properties also apply to labelled transition semantics, which provide finer notions of observation.

here, arbitrary 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 pre-administratively 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 [Wal92] 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. Furthermore, proofs of full abstraction are usually based on some operational correspondence between source terms and translations. In most cases, formulas resembling $\mathfrak{I}$ appear as lemmas for full abstraction.

For reasoning about the choice encoding's behavioral correctness, we shall be able to compare source terms S and their translations $[\![\, S \,]\!]$ directly; both encodings are endomorphic mappings where the target language $\mathbb{T}$ is a fragment of the source language $\mathbb{S}$ and where visible labels of source and target transitions are the same.

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.

5 A distinguishing example

In this section, we highlight the difference between the two choice encodings by comparing a particular source term with its both translations. We investigate the source term

$$S \;\equiv\; \overline{y_2}\langle z\rangle \;\mid\; N \qquad \textit{where} \qquad N \;\equiv\; y_1(x).P_1 \;+\; y_2(x).P_2$$

describing a binary choice in the presence of a single message matching the second of the branches, where P_1, P_2 are arbitrary target terms (i.e. not containing choices). Both,

$$S \not\approx \mathcal{C}[\![\, S \,]\!] \qquad \text{and} \qquad S \approx \mathcal{D}[\![\, S \,]\!]$$

hold, which together imply that the $\mathcal{C}$-encoding neither preserves nor reflects weak bisimulation (Lemmas 5.3.1 and 5.3.2).

5.1 Divergence-free encoding

The transition systems of S and $\mathcal{C}[\![\, S \,]\!]$ can be depicted as follows. Remember that input transitions are only considered in the context of arbitrary messages. Therefore, we only mention internal and output transitions since those are to be simulated literally. The dotted lines representing simulation relations are to be read from left to right; when the lines are vertical, simulation holds in both directions.

$$\begin{tikzcd}
\mathbb{S}: & N \arrow[d, dotted, no head, "\preccurlyeq\ \succcurlyeq"] & S \arrow[l, "\overline{y_2}\langle z\rangle"'] \arrow[rr, "\tau"] \arrow[d, dotted, no head, "\preccurlyeq\ \succcurlyeq"] \arrow[dr, dotted, no head, "\succcurlyeq"] & & P_2\{^z\!/_x\} \arrow[dl, dotted, no head, "\succcurlyeq"'] \arrow[d, dotted, no head, "\preccurlyeq\ \succcurlyeq"] \\
\mathbb{T}: & \mathcal{C}[\![\, N \,]\!] & \mathcal{C}[\![\, S \,]\!] \arrow[l, "\overline{y_2}\langle z\rangle"'] \arrow[r, "\tau"] & C \arrow[r, "\tau"] & C_2
\end{tikzcd}$$

$$\begin{aligned}
\text{where:}\quad \mathcal{C}[\![\, S \,]\!] &\equiv \overline{y_2}\langle z\rangle \mid \mathcal{C}[\![\, N \,]\!] \\
\mathcal{C}[\![\, N \,]\!] &\equiv (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{t}\rangle \mid \mathit{Init}_l\langle\, \mathcal{C}[\![\, y_1(x).P_1 \,]\!]\,\rangle \mid \mathit{Init}_l\langle\, \mathcal{C}[\![\, y_2(x).P_2 \,]\!]\,\rangle\ \Big) \\
C &\equiv (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{t}\rangle \mid \mathit{Init}_l\langle\, \mathcal{C}[\![\, y_1(x).P_1 \,]\!]\,\rangle \mid \mathit{Lock}_l\langle\, \mathcal{C}[\![\, y_2(x).P_2 \,]\!]\,\rangle\ \{^z\!/_x\}\ \Big) \\
C_2 &\equiv (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{f}\rangle \mid \mathit{Init}_l\langle\, \mathcal{C}[\![\, y_1(x).P_1 \,]\!]\,\rangle \mid \mathit{Exit}_l^{\boldsymbol{t}}\langle\, \mathcal{C}[\![\, y_1(x).P_1 \,]\!]\,\rangle\ \{^z\!/_x\}\ \Big) \\
&\equiv \underbrace{(\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{f}\rangle \mid \mathit{Init}_l\langle\, \mathcal{C}[\![\, y_1(x).P_1 \,]\!]\,\rangle\ \Big)}_{\approx\mathbf{0}\ \text{(consequence of Lemma 6.2.2)}} \mid \mathcal{C}[\![\, P_2 \,]\!]\{^z\!/_x\}
\end{aligned}$$

We may phrase the intermediate state C as having only *partially committed*: *one* of the branches will eventually be chosen, but, at that stage of commitment, it is not yet clear if it will be the activated branch or if it will be the competing branch waiting at y_1, 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 5.1.1 $S \not\approx \mathcal{C}[\![\, S \,]\!]$.

Nevertheless, 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.

5.2 Divergent encoding

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

$$\begin{tikzcd}
\mathbb{S}: & N \arrow[d, dotted, no head, "\approx"] & S \arrow[l, "\overline{y_2}\langle z\rangle"'] \arrow[rrr, "\tau"] \arrow[d, dotted, no head, "\approx"] \arrow[dr, dotted, no head, "\approx"] \arrow[drr, dotted, no head, "\approx"] & & & P_2\{^z\!/_x\} \arrow[d, dotted, no head, "\approx"] \\
\mathbb{T}: & \mathcal{D}[\![\, N \,]\!] & \mathcal{D}[\![\, S \,]\!] \arrow[l, "\overline{y_2}\langle z\rangle"'] \arrow[r, "\tau"] & D \arrow[r, "\tau"] & D' \arrow[r, "\tau"] \arrow[ll, bend left, "\tau"] & D_2
\end{tikzcd}$$

where (letting $\mathit{INIT}_i = \,!\, \mathit{Init}_l\langle\, \mathcal{D}[\![\, y_i(x).P_i \,]\!] \,\rangle$):

$$
\begin{array}{rcl}
\mathcal{D}[\![\, S \,]\!] & \equiv & \overline{y_2}\langle z\rangle \;\mid\; \mathcal{D}[\![\, N \,]\!] \\
\mathcal{D}[\![\, N \,]\!] & \equiv & (\boldsymbol{\nu} l)\Big(\; \bar{l}\langle \boldsymbol{t}\rangle \;\mid\; \mathit{INIT}_1 \;\mid\; \mathit{INIT}_2 \;\Big) \\
D & \equiv & (\boldsymbol{\nu} l)\Big(\; \bar{l}\langle \boldsymbol{t}\rangle \;\mid\; \mathit{INIT}_1 \;\mid\; \mathit{INIT}_2 \quad\mid\; \mathit{Lock}_l\langle\, \mathcal{D}[\![\, y_2(x).P_2 \,]\!] \,\rangle\;\; \{^z\!/\!_x\} \;\Big) \\
D' & \equiv & (\boldsymbol{\nu} l)\Big(\; \qquad \mid\; \mathit{INIT}_1 \;\mid\; \mathit{INIT}_2 \quad\mid\; \mathit{Exit}^{\boldsymbol{t}}_l\langle\, \mathcal{D}[\![\, y_2(x).P_2 \,]\!] \,\rangle\;\; \{^z\!/\!_x\} \;\Big) \\
 & \equiv & (\boldsymbol{\nu} l)\Big(\; \qquad \mid\; \mathit{INIT}_1 \;\mid\; \mathit{INIT}_2 \quad\mid\; (\mathcal{D}[\![\, P_2 \,]\!]\{^z\!/\!_x\} \mid \bar{l}\langle \boldsymbol{f}\rangle) \oplus (\overline{y_2}\langle z\rangle \mid \bar{l}\langle \boldsymbol{t}\rangle) \;\Big) \\
D_2 & \equiv & \underbrace{(\boldsymbol{\nu} l)\Big(\; \bar{l}\langle \boldsymbol{f}\rangle \;\mid\; \mathit{INIT}_1 \;\mid\; \mathit{INIT}_2 \;\Big)}_{\approx \mathbf{0}\ \text{(consequence of Lemma 6.6.1)}} \;\mid\; \mathcal{D}[\![\, P_2 \,]\!]\{^z\!/\!_x\}
\end{array}
$$

The undo-arrow from D' back to $\mathcal{D}[\![\, S \,]\!]$ is essential to prove 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. Consequently:

Proposition 5.2.1 $S \approx \mathcal{D}[\![\, S \,]\!]$.

5.3 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 $\mathcal{D}[\![\ ]\!]$ act as an identity on target terms, 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 5.3.1 *$\mathcal{C}[\![\, S_1 \,]\!] \approx \mathcal{C}[\![\, S_2 \,]\!]$ 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 $S_2 \not\approx S$. $\square$

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

Lemma 5.3.2 *$S_1 \approx S_2$ 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 $S_1 \approx S_2$. Assume, 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 5.1.1. $\square$

6 Correctness by decoding

In the remainder of the paper, we prove the correctness of the $\mathcal{C}$-encoding up to coupled simulation (Subsections 6.2–6.5) and sketch the corresponding proof for the $\mathcal{D}$-encoding up to weak bisimulation (Subsection 6.6). We also include a proof that the $\mathcal{C}$-encoding is divergence-free, which one might consider as part of its correctness property (Subsection 6.7).

6.1 Proof outline

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 $\mathcal{C}$-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. Intermediate states are precisely 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. Therefore, we introduce *annotated* source terms (Subsection 6.2) 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 $\mathcal{C}$-encoding:

Factorization (Subsection 6.3) 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.

$$\begin{tikzcd}
\mathbb{S} \arrow[r, "\mathcal{A}[\![\ ]\!]"] \arrow[dr, "\mathcal{C}[\![\ ]\!]"'] & \mathbb{A} \arrow[d, "\mathcal{F}[\![\ ]\!]"] \\
 & \mathbb{T}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathbb{S} & \mathbb{A} \arrow[l, shift right, "\mathcal{U}_\sharp[\![\ ]\!]"'] \arrow[l, shift left, "\mathcal{U}_\flat[\![\ ]\!]"]
\end{tikzcd}$$

Decoding (Subsection 6.4) 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.

The factorization and the decodings enjoy several nice properties:

1. $\mathcal{F}$ is a strong bisimulation between abbreviations and target terms ($\mathbb{A} \times \mathbb{T}$).
2. $(\mathcal{U}_\flat, \mathcal{U}_\sharp)$ is a coupled simulation between abbreviations and source terms ($\mathbb{A} \times \mathbb{S}$).

Those can be combined to provide a coupled simulation $(\mathcal{U}_\flat^{-1}\mathcal{F}, \mathcal{U}_\sharp^{-1}\mathcal{F})$ on source and target terms ($\mathbb{S} \times \mathbb{T}$). The observation that every source term S and its translation $\mathcal{C}[\![\, S \,]\!]$ are related by this coupled simulation concludes the proof of coupled-simulation-correctness of the $\mathcal{C}$-encoding (Subsection 6.5).

6.2 Annotated choice

This subsection introduces an annotated variant of choice, which provides abbreviations for all derivatives of translations. Its shape reveals the high-level structure, but it exhibits low-level operational behavior that is defined by an operational semantics. The attached annotations record essential information about the low-level derivation history.

Auxiliary notation. We use $A - a$ to denote the removal $A \setminus \{a\}$ of an element a from a set A. $A + a$ denotes the union $A \cup \{a\}$ where $a \notin A$. A partial function $\rho : C \rightharpoonup D$ is defined on $\mathit{dom}(\rho) \subseteq C$. For $c \in C - \mathit{dom}(\rho)$ and $d \in D$, the partial function $\rho + (c \mapsto d)$ denotes an extension of ρ with value d for input c. By $\rho - c$, we mean $\rho \restriction_{\mathit{dom}(\rho) - c}$ for $c \in C$. The empty set $\emptyset$ also denotes the everywhere-undefined function.

According to the definition in Section 3, the individual branches of a choice may inhabit one of basically three different states: *Init*, *Lock*, or *Exit*. In fact, the state of a branch is completely determined (e.g. *Exit* will develop to either *Commit* or *Abort*) 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 abstract representation of that information for a J-indexed choice, we use

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

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 $V(j)$ 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 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, the state of each individual branch can be retrieved from the annotations V and B. Branch $k \in J$ is in

- init-state if $k \in J \setminus (V \cup B)$
- lock-state if $k \in V$
- exit-state if $k \in B$,

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

Table 3: Early transition semantics for annotated choice

where, by abuse of notation, we write $k \in V$ to mean $k \in \mathsf{dom}(V)$.

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

Definition 6.2.1 (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 $V : J \rightharpoonup \mathbf{V}$ and $B \subseteq J$ with $B \cap \mathsf{dom}(V) = \emptyset$. The notations*

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

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

Annotated choice is given the operational semantics in Table 3. The dynamics of annotated choice mimic precisely the behavior of the intended low-level process. *READ* allows a branch k in init-state ($k \in J \setminus (V \cup B)$) to optimistically consume a message.[2] If the choice is not yet resolved ($B = \emptyset$), *COMMIT* specifies that an arbitrary branch k in lock-state ($k \in V$) can immediately commit and trigger its continuation process P_k. After the choice is resolved ($B \neq \emptyset$), *ABORT* allows branches k in lock-state ($k \in V$) to release their consumed messages. Intuitively, by reading the lock, a branch immediately leaves the choice system and exits. Therefore, annotated choice only contains branches in either init- or lock-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 exit-state ($B = J$).

Committed choice exhibits a particularly interesting property: its branches in lock-state already have consumed a message which they will return after recognizing, by internally reading the lock, that the choice is already committed; its branches in init-state are still

[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. Nevertheless, the further development in this paper does not depend on this decision. Note also that early and late bisimulation coincide in our setting [ACS96].

waiting for values to be consumed and resent. Processes with such receive-and-resend behavior are (weak asynchronous) bisimilar to $\mathbf{0}$ and are also called *identity receptors* [HT92]. In fact, a stronger property holds:

Lemma 6.2.2 *Let $B \neq \emptyset$. Then* $\big(\sum_{j \in J} R_j\big)_B^V \gtrsim \prod_{j \in V} \overline{y_j}\langle V(j)\rangle$.

Proof: We define a relation by composing both the annotated choice and the messages with arbitrary other messages and prove that it is an expansion (see Appendix A.3.1). $\square$

Corollary 6.2.3 (Inertness) *Let $B \neq \emptyset$. Then* $\big(\sum_{j \in J} R_j\big)_B^{\emptyset} \gtrsim \mathbf{0}$.

From an asynchronous observer's point of view, committed choice behaves exactly like the composition of the messages that are held by branches in lock-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.

6.3 Factorization

Since the purpose of annotated choice is keeping track of which low-level actions belong to the same high-level choice, we introduce a language $\mathbb{P}^+$ of *possibly annotated processes*, which contains the source $\mathbb{S}$ as a sublanguage. It is generated by the following grammar:

$$
\begin{array}{rcl}
R & ::= & y(x).P \\
P & ::= & \mathbf{0} \;\;\big|\;\; (\boldsymbol{\nu} x)\, P \;\;\big|\;\; P|P \;\;\big|\;\; \mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ P \;\;\big| \\
 & & \overline{y}\langle z\rangle \;\;\big|\;\; R \;\;\big|\;\; !\,R \;\;\big|\;\; \sum_{j\in J} R_j \;\;\big|\;\; \big(\sum_{j\in J} R_j\big)_B^V
\end{array}
$$

As for the language $\mathbb{P}$, names x range over $\mathbf{N}$, and y, z over $\mathbf{V}$ which includes the special names $\boldsymbol{t}$ and $\boldsymbol{f}$. The operational semantics of $\mathbb{P}^+$ is provided by the rules in Table 1 and Table 3.

We now introduce the remaining components for the factorization diagram: an *annotation* encoding $\mathcal{A}[\![\ ]\!]$, a *flattening* encoding $\mathcal{F}[\![\ ]\!]$, and the intermediate sublanguage $\mathbb{A}$. We write $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{F}$ for the encoding functions considered as relations.

$$
\begin{tikzcd}
\mathbb{S} \arrow[r, "\mathcal{A}[\![\ ]\!]"] \arrow[dr, "\mathcal{C}[\![\ ]\!]"'] & \mathbb{A} \arrow[d, "\mathcal{F}[\![\ ]\!]"] \\
 & \mathbb{T}
\end{tikzcd}
\qquad \textit{with} \quad
\begin{array}{c}
\mathbb{T} \subset \mathbb{S} \subset \mathbb{P}^+ \\
\mathbb{T} \subset \mathbb{A} \subset \mathbb{P}^+
\end{array}
\quad \textit{and} \quad
\begin{array}{c}
\mathbb{S} \not\subseteq \mathbb{A} \\
\mathbb{A} \not\subseteq \mathbb{S}
\end{array}
$$

Annotation. The encoding $\mathcal{A}[\![\ ]\!] : \mathbb{S} \to \mathbb{P}^+$ acts homomorphically on every constructor except for choice according to the scheme in Section 3. The latter case is given by

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

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

Proof: See Appendix A.3.4. □

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. The basic syntactic properties 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 6.4.6 and 6.4.8), 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 $\mathbb{P}^+$ that satisfy the above syntactic requirements; Appendix A.2 contains an inductive grammar that generates $\mathbb{A}$. 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.

Lemma 6.3.2 (Transition-closure of $\mathbb{A}$) *For all $A \in \mathbb{A}$, if $A \xrightarrow{\mu} A'$, then $A' \in \mathbb{A}$.*

Proof: By inspection of the rules in Table 3. □

Flattening. The encoding $\mathcal{F}[\![\ ]\!] : \mathbb{A} \to \mathbb{T}$ acts homomorphically on every constructor but annotated choice according to the scheme in Section 3. Let $\boldsymbol{b}$ be $\boldsymbol{t}$ if $B = \emptyset$ and $\boldsymbol{f}$ otherwise. For annotated choice, the translation

$$\mathcal{F}[\![\big(\sum_{j\in J} R_j\big)^V_B]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu} l) \Big(\ \bar{l}\langle \boldsymbol{b}\rangle \ \mid \prod_{j\in J\setminus(V\cup B)} Init_l\langle\, \mathcal{F}[\![R_j]\!] \,\rangle \ \mid \ \prod_{j\in V} Lock_l\langle\, \mathcal{F}[\![R_j]\!] \,\rangle\{^{V(j)}\!/_x\} \ \Big)$$

expands the abbreviations into the intended target term by following the semantic rules in Table 3. Branches in lock-state are those which carry values (therefore $j \in V$); the substitution $\{^{V(j)}\!/_x\}$ replaces the input variable in the continuation process P_j with the corresponding value. Branches in init-state must neither currently carry values ($\notin V$) nor have accessed the lock after reading values ($j \notin B$).

Lemma 6.3.3 (Factorization) *1.* $\mathcal{F}[\![\ ]\!] \circ \mathcal{A}[\![\ ]\!] = \mathcal{C}[\![\ ]\!]$*.*

2. $\mathcal{F}[\![\ ]\!]$ *is surjective.*

3. For all $S \in \mathbb{S}$*,* $\mathcal{A}[\![\, S\sigma \,]\!] = \mathcal{A}[\![\, S \,]\!]\sigma$*, and 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. Since $\mathcal{F}[\![\ ]\!]$ is a homomorphic identity on $\mathbb{T}$, and $\mathbb{T} \subset \mathbb{A}$.
3. Straightforward. Neither $\mathcal{A}[\![\ ]\!]$ nor $\mathcal{F}[\![\ ]\!]$ erase names. Free (bound) occurrences of names of terms correspond to free (bound) occurrences in their translations. □

The most important property of the factorization is that the semantics of annotated choice (cf Table 3) precisely mirrors the behavior of the original translations and their derivatives.

Proposition 6.3.4 (Semantic correctness) $\mathcal{F}$ *is a strong bisimulation.*

Proof: See Appendix A.3.3. □

Thus, we may 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 much more structure, in particular concerning partially committed derivatives, which is heavily exploited in Section 6.4.

6.4 Decoding derivatives of translations

We want to construct a coupled simulation 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 Section 3. 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,

$$
\begin{array}{lll}
\textit{initial}: & \mathcal{U}[\![\, (\sum_{j\in J} R_j)^{\emptyset}_{\emptyset} \,]\!] \stackrel{\text{def}}{=} \sum_{j\in J} \mathcal{U}[\![\, R_j \,]\!] & \\
\textit{committed}: & \mathcal{U}[\![\, (\sum_{j\in J} R_j)^{V}_{B} \,]\!] \stackrel{\text{def}}{=} \prod_{j\in V} \overline{y_j}\langle V(j)\rangle & \textit{if } B \neq \emptyset
\end{array}
$$

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.

$$
\textit{partial}: \quad \mathcal{U}_\flat[\![\, (\sum_{j\in J} R_j)^{V}_{\emptyset} \,]\!] \stackrel{\text{def}}{=} \prod_{j\in V} \overline{y_j}\langle V(j)\rangle \mid \sum_{j\in J} \mathcal{U}_\flat[\![\, R_j \,]\!] \qquad \textit{if } V \neq \emptyset
$$

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 $\textit{take}(V)$ select an arbitrary element of V. Then,

$$
\textit{partial}: \quad \mathcal{U}_\sharp[\![\, (\sum_{j\in J} R_j)^{V}_{\emptyset} \,]\!] \stackrel{\text{def}}{=} \prod_{j\in V-k} \overline{y_j}\langle V(j)\rangle \mid \mathcal{U}_\sharp[\![\, P_k \,]\!]\{^{V(k)}\!/_{x}\} \qquad \textit{if } V \neq \emptyset,
$$

where $k = \textit{take}(V)$, maps to the source-level commitment to the selected branch.

Lemma 6.4.1 (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), and straightforward by induction (3). $\square$

We will prove that $(\mathcal{U}_\flat, \mathcal{U}_\sharp)$ is a coupled asynchronous simulation on $\mathbb{A} \times \mathbb{S}$ (Proposition 6.4.9). This requires, in particular, that $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp^{-1}$ are weak asynchronous simulations. We show first that $\mathcal{U}_\flat[\![A]\!]$ *simulates* A (i.e. $A \preccurlyeq \mathcal{U}_\flat[\![A]\!]$) for all $A \in \mathbb{A}$.

Lemma 6.4.2 *$\mathcal{U}_\flat$ is a weak simulation.*

Proof: See Appendix A.3.5. □

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

Proof: See Appendix A.3.6. □

In addition, the weak simulations $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp^{-1}$ satisfy further properties that will be useful for the discussion of divergence in Subsection 6.7.

Lemma 6.4.4 *1. $\mathcal{U}_\flat$ is strict.*

2. $\mathcal{U}_\sharp^{-1}$ is faithful.

3. Both $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp^{-1}$ are progressing.

Proof: Both the strictness of $\mathcal{U}_\flat$ and the faithfulness of $\mathcal{U}_\sharp^{-1}$ follow from previous proofs: For $\mathcal{U}_\flat$, proof A.3.5 shows that some τ-steps of A may be simulated trivially by $\mathcal{U}_\flat[\![A]\!]$:

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

For $\mathcal{U}_\sharp^{-1}$, proof A.3.6 shows that no τ-step of $\mathcal{U}_\sharp[\![A]\!]$ may be suppressed by A:

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

The faithfulness of $\mathcal{U}_\sharp^{-1}$ directly implies that it is progressing.

However, we have not yet proven that $\mathcal{U}_\flat$ is also 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 only have to argue that there is an upper bound k_A for the number of subsequent cases where the τ-step may be simulated trivially such that for $n > k_A$:

$$A = A_0 \xrightarrow{\tau} A_1 \xrightarrow{\tau} A_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} A_n \qquad \text{implies} \qquad \mathcal{U}_\flat[\![A]\!] \xrightarrow{\tau}{}^{+} \mathcal{U}_\flat[\![A_n]\!].$$

The details of that argument can be found in Appendix A.3.7. □

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 6.4.5 *Let $A \in \mathbb{A}$ be fully committed. Then $\mathcal{U}_\sharp[\![\, A \,]\!] = \mathcal{U}_\flat[\![\, A \,]\!]$.*

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 6.3 and Appendix A.2.

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

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

Lemma 6.4.6 ($\mathbb{S}$-coupling) *For all $A \in \mathbb{A}$, $\mathcal{U}_\flat[\![\, A \,]\!] \Rightarrow \mathcal{U}_\sharp[\![\, A \,]\!]$.*

Proof: By structural induction and analysis of the annotations on occurrences of choice in $A \in \mathbb{A}$. Transitions at the source-level are derived by committing those source-level choices that correspond to unguarded occurrences of choice in A: for each of them, according to the selection of an active branch in the definition of $\mathcal{U}_\sharp[\![\ ]\!]$, we take a branch and derive one internal transition for $\mathcal{U}_\flat[\![\, A \,]\!]$ using the rule *C-INP* (more details are given in Appendix A.3.8). $\square$

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

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

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 6.4.7 *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: By structural induction and analysis of the annotations of occurrences of choice in $A \in \mathbb{A}$. The term A' is constructed by committing every unguarded occurrence of choice: for each occurrence, according to the selection of an active branch in the definition of $\mathcal{U}_\sharp[\![\ ]\!]$, we take a branch and derive an internal transition for A using the rule *COMMIT*. The details can be found in Appendix A.3.9. $\square$

Lemma 6.4.8 ($\mathbb{A}$-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 6.4.7. Thus, we know that $\mathcal{U}_\sharp[\![\, A \,]\!] = \mathcal{U}_\sharp[\![\, A' \,]\!]$ and, since A' is fully committed, we also know (Fact 6.4.5) 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 6.4.9 *$(\mathcal{U}_\flat, \mathcal{U}_\sharp)$ is a coupled simulation.*

Proof: By Lemmas 6.4.2 and 6.4.3, we know that $(\mathcal{U}_\flat, \mathcal{U}_\sharp)$ is a mutual simulation. By Lemmas 6.4.6 and 6.4.8, we have the necessary coupling between $\mathcal{U}_\flat$ and $\mathcal{U}_\sharp$. $\square$

Since the definition of coupled simulation is symmetric, we also have:

Corollary 6.4.10 *$(\mathcal{U}_\sharp^{-1}, \mathcal{U}_\flat^{-1})$ is a coupled simulation.*

Before ending this section, we show how the decoding functions may be used to provide a notably sharp operational soundness 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:

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

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 $\mathcal{U}_\sharp[\![A]\!]$, as proved by Proposition 6.4.6 ($\mathbb{S}$-coupling). 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 translations $\mathcal{A}[\![S]\!]$. It may be written as $S \rightleftarrows \mathcal{A}[\![S]\!]$ for every $S \in \mathbb{S}$, since it can be shown 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 $\mathcal{C}[\![\ ]\!]$. Those results are assembled in the following subsection.

6.5 Main result

In this subsection, we establish a coupled simulation between source terms and their $\mathcal{C}$-translations by exploiting the results for the $\mathcal{A}$-encoding in the previous sections. Reasoning about the annotated versions of choice allowed us to use their high-level structure for the decoding functions. We argued that we could concentrate on the annotated language $\mathbb{A}$, since its flattening $\mathcal{F}$ expanded the abbreviations correctly into target terms. In order to combine those ideas, let the simulations $\mathfrak{C}$ (completeness) and $\mathfrak{S}^{-1}$ (soundness) be defined by

$$\mathfrak{C} \stackrel{\text{def}}{=} \mathcal{U}_\sharp^{-1}\mathcal{F} \quad \subset \mathbb{S} \times \mathbb{T} \qquad \text{and} \qquad \mathfrak{S} \stackrel{\text{def}}{=} \mathcal{U}_\flat^{-1}\mathcal{F} \quad \subset \mathbb{S} \times \mathbb{T}.$$

The results for abbreviated target terms in $\mathbb{A}$ now carry over to target terms in $\mathbb{T}$.

Theorem 6.5.1 *($\mathfrak{C}, \mathfrak{S}$) is a coupled simulation.*

Proof: By Corollary 6.4.10, Proposition 6.3.4, and Lemma 2.5.3. □

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 $\mathcal{C}$-translations.

Lemma 6.5.2 *For all $S \in \mathbb{S}$, we have $(S, \mathcal{C}[\![S]\!]) \in \mathfrak{C} \cap \mathfrak{S}$.*

Proof: By the syntactic adequacy lemmas (6.4.1 and 6.3.3), we know that, for all $S \in \mathbb{S}$, the translation $\mathcal{A}[\![S]\!]$ yields a witness for $(S, \mathcal{C}[\![S]\!])$ being contained in both $\mathfrak{C}$ and $\mathfrak{S}$. □

Thus, the $\mathcal{C}$-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 6.5.3 (Correctness of $\mathcal{C}$) *For all $S \in \mathbb{S}$, we have $S \rightleftarrows \mathcal{C}[\![\, S \,]\!]$.*

Proof: By Theorem 6.5.1 and Lemma 6.5.2. $\square$

Corollary 6.5.4 (Full abstraction) *For all $S_1, S_2 \in \mathbb{S}$, $\quad S_1 \rightleftarrows S_2$ iff $\mathcal{C}[\![\, S_1 \,]\!] \rightleftarrows \mathcal{C}[\![\, S_2 \,]\!]$.*

Proof: By transitivity. $\square$

Note that the $\mathcal{C}$-encoding is not fully abstract up to weak bisimulation, as proven in Section 5.

Theorem 6.5.3 is a powerful statement of the correctness of the $\mathcal{C}$-encoding: source terms and their $\mathcal{C}$-translations are semantically equivalent with respect to asynchronous coupled simulation, so they have the same externally visible asynchronous branching behavior in every term context within $\mathbb{S}$.

6.6 Correctness of the divergent protocol

The proof for the divergent choice encoding follows the outline of Sections 6.2 to 6.5. In contrast to the divergence-free encoding, the $\mathcal{D}$-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:

$$
\begin{tikzcd}
\mathbb{S} \arrow[r, shift left, leftarrow, "\mathcal{U}[\![\,]\!]"] \arrow[r, shift right, "\mathcal{A}[\![\,]\!]"'] \arrow[dr, "\mathcal{C}[\![\,]\!]"'] & \mathbb{A} \arrow[d, "\mathcal{F}[\![\,]\!]"] \\
& \mathbb{T}
\end{tikzcd}
$$

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 taken by an activated branch, the choice is not (yet) resolved, since this branch may decide to undo its activity.

Let J be some indexing set. Let $V \rightharpoonup \mathbb{N}^{\mathbf{V}}$ be a partial function mapping indices to multisets of values and let $V + (k, z)$ and $V - (k, z)$ denote appropriate extensions and

READ:	$\big(\sum_{j\in J} R_j\big)^{V}_{\boldsymbol{b}}$	$\xrightarrow{y_k\langle z\rangle}$	$\big(\sum_{j\in J} R_j\big)^{V+(k,z)}_{\boldsymbol{b}}$	*for some* $k \in J$
TRY:	$\big(\sum_{j\in J} R_j\big)^{V}_{\boldsymbol{t}}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{V}_{(k,z)}$	*for some* $z \in V(k)$
COMMIT:	$\big(\sum_{j\in J} R_j\big)^{V}_{(k,z)}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{V-(k,z)}_{\boldsymbol{f}} \mid P_k\{{}^{z}\!/\!_{x}\}$	
UNDO:	$\big(\sum_{j\in J} R_j\big)^{V}_{(k,z)}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{V-(k,z)}_{\boldsymbol{t}} \mid \overline{y_k}\langle z\rangle$	
ABORT:	$\big(\sum_{j\in J} R_j\big)^{V}_{\boldsymbol{f}}$	$\xrightarrow{\tau}$	$\big(\sum_{j\in J} R_j\big)^{V-(k,z)}_{\boldsymbol{f}} \mid \overline{y_k}\langle z\rangle$	*for some* $z \in V(k)$

Table 4: Transition semantics for annotated divergent choice

removals of single index-value pairs. Let $\boldsymbol{b} \in \{\boldsymbol{t}, \boldsymbol{f}\} \cup \{\, (k,z) \in J \times \mathbf{V} \mid z \in V(k) \,\}$ denote either the state of the lock or a single index-value pair. The annotated divergent choice

$$\big(\sum_{j\in J} R_j\big)^{V}_{\boldsymbol{b}}$$

is given the operational semantics in Table 4.

Lemma 6.6.1 (Inertness) $\big(\sum_{j\in J} R_j\big)^{V}_{\boldsymbol{f}} \approx \prod_{j\in V} \overline{y_j}\langle V(j)\rangle.$

Factorization. The *annotation* translation is almost the same as in Section 6.3. Here, we have to initialize the lock-information with $\boldsymbol{t}$.

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

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

$$\mathcal{F}[\![\, \big(\sum_{j\in J} R_j\big)^{V}_{\boldsymbol{b}} \,]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu} l)\Big(\; T \;\mid\; \prod_{j\in J} \mathit{Init}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle \;\mid\; \prod_{(j,z)\in V-\boldsymbol{b}} \mathit{Lock}_l\langle\, \mathcal{F}[\![\, R_j \,]\!] \,\rangle\{{}^{z}\!/\!_{x}\} \;\Big)$$

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

Proposition 6.6.2 $\mathcal{F}$ *is a strong bisimulation.*

Decoding. The decoding function must also take care of "hesitating" branches.

$$\mathcal{U}[\![\, (\sum_{j\in J} R_j)^V_{\boldsymbol{b}} \,]\!] \stackrel{\mathrm{def}}{=} \prod_{j\in V} \overline{y_j}\langle V(j)\rangle \quad | \quad S$$

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

Proposition 6.6.3 *$\mathcal{U}$ is a weak bisimulation.*

Theorem 6.6.4 *$\mathcal{U}^{-1}\mathcal{F}$ is a weak bisimulation.*

Theorem 6.6.5 (Correctness of $\mathcal{D}$) *For all $S \in \mathbb{S}$, we have $S \approx \mathcal{D}[\![\, S \,]\!]$.*

Corollary 6.6.6 (Full abstraction) *For all $S_1, S_2 \in \mathbb{S}$, $\quad S_1 \approx S_2$ iff $\mathcal{D}[\![\, S_1 \,]\!] \approx \mathcal{D}[\![\, S_2 \,]\!]$.*

For every annotated choice term with at least one value, divergent computations, i.e. τ-loops, can easily be derived. For example,

$$(\sum_{j\in J} R_j)^V_{\boldsymbol{t}} \xrightarrow{\tau} (\sum_{j\in J} R_j)^V_{(k,z)} \xrightarrow{\tau} (\sum_{j\in J} R_j)^{V-(k,z)}_{\boldsymbol{t}} \mid \overline{y_k}\langle z\rangle \xrightarrow{\tau} (\sum_{j\in J} R_j)^V_{\boldsymbol{t}}$$

where $V(k) = z$, is infinitely often trying, undoing, reading, ...

6.7 Divergence

In section 3, we have claimed that our two choice encodings differ in their divergence-behavior. Our results in the previous subsections do not provide any rigorous justification of this, since the definition of weak simulation ignores divergence, as we discussed in Section 2.3.

An encoding is 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 a derivative S' of S, i.e. $T' \Uparrow$ implies $S' \Uparrow$. Whereas it is simple to show that the $\mathcal{D}$-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), as we did in the previous section, our claim that the $\mathcal{C}$-encoding is divergence-free requires a proof (cf Theorem 6.7.4).

As before, we carry out the proof by exploiting the rich structure provided by annotated terms. The following lemmas draw the connection between the divergence of annotated terms and their flattenings, which allows us to conclude that it suffices to study the divergence properties of $\mathcal{A}[\![\ ]\!]$. Since $\mathcal{F}$ is a strong bisimulation, an immediate consequence is:

Fact 6.7.1 *For all $A \in \mathbb{A}$, we have $A \Uparrow$ iff $\mathcal{F}[\![\, A \,]\!] \Uparrow$.*

Lemma 6.7.2 *$\mathcal{C}[\![\ ]\!]$ is divergence-free iff $\mathcal{A}[\![\ ]\!]$ is divergence-free.*

Proof: Since $\mathcal{F}$ is a strong bisimulation, it precisely mirrors all derivations of $\mathcal{C}$-translations at the $\mathbb{A}$-level. Furthermore, it preserves divergence in both simulation directions. $\square$

Thus, it suffices to prove that $\mathcal{A}[\![\ ]\!]$ is divergence-free, i.e. that infinite τ-sequences of derivatives A' of $\mathcal{A}[\![\,S\,]\!]$ are mirrored by infinite τ-sequences of derivatives S' of S at the source level. We proceed by exhibiting a divergence-preserving simulation containing all pairs of potentially diverging corresponding derivatives (A', S').

Again, we exploit our decoding functions. In particular, we have already shown that $\mathcal{U}_\flat$ is a progressing weak simulation (Lemma 6.4.4) and, furthermore, it contains all required pairs.

Proposition 6.7.3 $\mathcal{A}[\![\ ]\!]$ *is divergence-free.*

Proof: For all $S \in \mathbb{S}$ and $\mathcal{A}[\![\,S\,]\!] \xRightarrow{a_1 \cdots a_n} A'$, we have $(\mathcal{U}_\flat \circ \mathcal{A})[\![\,S\,]\!] = S \xRightarrow{a_1 \cdots a_n} \mathcal{U}_\flat[\![\,A'\,]\!] =: S'$ by weak simulation, and $A' \Uparrow$ implies $\mathcal{U}_\flat[\![\,A'\,]\!] \Uparrow$ since $\mathcal{U}_\flat$ is progressing. $\square$

Theorem 6.7.4 $\mathcal{C}[\![\ ]\!]$ *is divergence-free.*

Proof: By Proposition 6.7.3 and Lemma 6.7.2. $\square$

Another way of formulating the result is by stating that the soundness simulation $\mathfrak{S}^{-1}$ is progressing. This is in fact true since $\mathfrak{S}^{-1}$ is defined as $\mathcal{F}^{-1}\mathcal{U}_\flat$ as the composition of two progressing simulations.

7 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 in more detail.

Correctness: For both choice encodings, we have provided a framework that allowed us to compare source terms and their translations directly. Thereby, we could use a correctness notion that is stronger than the usual full abstraction, which here comes up as a simple corollary. The strength of our correctness result 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 (note that 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.

Asynchrony: For both encodings, their correctness proofs cannot be built upon standard (i.e. synchronous) notions of simulation. The reason is the inherent asynchrony of the 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 already correspond 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 up 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 capturing precisely 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, consequently, a proper setting for the definition of decodings.

According to the soundness interpretation of the correctness results for the $\mathcal{A}$-encoding in Subsection 6.4, 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 $S \in \mathbb{S}$,

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

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 of [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 by using the chemical abstract machine framework for the semantics of Facile [LT95].

In the CCS-setting of [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 modelled 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 [Mil91], 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 [San95c], 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 [Par95].

Much more work has been done on the compilation of whole languages into process calculi, exploring both semantics and expressivity. 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 [Mil90, San92, Lav93, San94a, Tho95, San95a, ALT95, Ode95b, Nie96], data types and other sequential programming constructs [Mil89, Mil91, Wal91b, Ode95a], from object-oriented languages into process calculi [Vaa90, Wal91a, Wal92, Wal93, Jon93, Wal94, PT95], from logic programming languages into the π-calculus [Ros92, 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 [PT96]. 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 faithful and strict, respectively, and both 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 and, furthermore, such that one of the component simulations is faithful while the other is strict. Another idea that arises immediately is to enhance coupled simulation with the requirement that the two simulations shall be progressing. In fact, $(\mathfrak{C}, \mathfrak{S})$ is such a *progressing coupled simulation* as the results in Section 6 show.

Putting all observations together, $(\mathfrak{C}, \mathfrak{S})$ would satisfy the requirements of *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 [ACS96], 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 (labelled) 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 [MS92, HY93], could be especially useful with respect to encodings between different calculi, since it rests on more 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.

Finally, we are interested in more sophisticated divergence-free choice encodings as they are used in the Pict language, especially with respect to efficiency and garbage collection issues. Further variants might address *events* [Rep92] or mixed guarded choice [Kna93]. All of these encodings have in common that they require channel manager processes in order to run more complicated protocols. Therefore, we cannot expect to be able to compare source terms and target terms directly. Furthermore, a correctness result will have to take into account that translations may only behave well in contexts that respect the protocol which is expected for the free names of the translation. Techniques like "firewalls" [FG96] might be necessary to protect translations from hostile contexts.

Acknowledgements

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 5.3.2. 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 Appendix

A.1 Choice encoding without primitive booleans

We use polyadic π-calculus, here, and instead of booleans we use a standard protocol for encoding their behavior [Mil91, Pie96] adapted to the needs of the example.

$$
\begin{array}{llll}
\mathcal{B}[\![\sum_{j\in J} y_j(\tilde{x}).P_j]\!] \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu} l) & \Big(\; l(t,f).\bar{t}\langle\rangle & & \\
& \Big(\; \prod_{j\in J} y_j(\tilde{x}).(\boldsymbol{\nu} t,f\,) & (\; \bar{l}\langle t,f\rangle & \text{(if)} \\
& & \mid t()\,.\,(\; \mathcal{B}[\![P_j]\!] \mid l(t,f).\overline{f}\langle\rangle\;) & \text{(then)} \\
& & \mid f()\,.\,(\; \overline{y_j}\langle\tilde{x}\rangle \mid l(t,f).\overline{f}\langle\rangle\;)\;) \;\Big) & \text{(else)}
\end{array}
$$

Note that only "local" booleans, i.e. those which reside always inside a restriction on their access name (cf. the definition of the sublanguage $\mathbb{S}$), are needed for the encoding. Note also that signals on f are not available unless the choice has been resolved, i.e. unless at least one branch has succeeded in receiving a message on its local t.

In contrast to $\mathcal{C}[\![\;]\!]$, the $\mathcal{B}$-encoding causes one additional τ-step for each interrogation of the lock. Furthermore, whereas in the $\mathcal{C}$-encoding branches are syntactically aborted, here they have to be explicitly dealt with, e.g. by structural laws

$$
\begin{array}{rrcl}
\textbf{garbage collection} & (\boldsymbol{\nu} y)\,(\overline{y}\langle\tilde{z}\rangle) & \equiv & \mathbf{0} \\
& (\boldsymbol{\nu} y)\,(y(\tilde{x}).P) & \equiv & \mathbf{0}
\end{array}
$$

which hold in the case of the π-calculus for every interleaving behavioral equivalence. The rules are applicable in the above example since the names t, f that trigger the encoding of continuation processes are always used at most once. In particular, for each such pair only one of the components will ever be used. This can be derived from the corresponding expressions.

A.2 Grammar for the intermediate language

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

$$
\begin{array}{lcl}
G & ::= & y(x).I \\
I & ::= & \overline{y}\langle z\rangle \;\mid\; G \;\mid\; !\,G \;\mid\; \big(\sum_{j\in J} G_j\big)^{\emptyset}_{\emptyset} \\
 & \mid & \mathbf{0} \;\mid\; (\boldsymbol{\nu} y)\,I \;\mid\; I|I \;\mid\; \mathsf{if}\ b\ \mathsf{then}\ I\ \mathsf{else}\ I
\end{array}
$$

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:

$$\begin{array}{rcl} R & ::= & y(x).I \\ A & ::= & \overline{y}\langle z\rangle \;\;\big|\;\; R \;\;\big|\;\; !\,R \;\;\big|\;\; \big(\sum_{j\in J} R_j\big)_B^V \\ & \big| & \mathbf{0} \;\;\big|\;\; (\boldsymbol{\nu} y)\,A \;\;\big|\;\; A|A \;\;\big|\;\; \mathsf{if}\ b\ \mathsf{then}\ A\ \mathsf{else}\ A \end{array}$$

where $V : J \rightharpoonup \mathbf{V}$ and $B \subseteq J$ for arbitrary index set J, as usual. The set of processes A, as generated by the above grammar, and the language $\mathbb{A}$, as introduced in Section 6.3, coincide since the syntactic requirements are directly expressed within the two-level definition.

A.3 Proofs in more detail

Let $\sigma_k = \{^{V(k)}/_x\}$ and $M_k = \overline{y_k}\langle V(k)\rangle$ for $k \in V$ be abbreviations in the context of annotations.

A.3.1 Proof of Lemma 6.2.2: Committed choice

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

$$\mathcal{R} \stackrel{\text{def}}{=} \Big\{ \Big(\underbrace{M \mid \big(\sum_{j\in J} R_j\big)_B^V}_{\text{LHS}} \,,\; \underbrace{M \mid \prod_{j\in V} M_j}_{\text{RHS}} \Big) \;\Big|\; \big(\sum_{j\in J} R_j\big)_B^V \in \mathbb{A} \Big\}$$

is an (asynchronous) 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

$$\text{LHS} = M \mid \big(\sum_{j\in J} R_j\big)_B^V \xrightarrow{\tau}\equiv M \mid M_k \mid \big(\sum_{j\in J} R_j\big)_{B+k}^{V-k} =: \text{LHS}'$$

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

$$M \mid M_k \mid \prod_{k\neq j\in V} M_j = M \mid \prod_{j\in V} M_j$$

coincides with RHS.

case *READ* For $M \equiv N|\overline{y}\langle z\rangle$ and $k \in J\backslash(V \cup B)$ such that $y = y_k$, let $V' = V+(k \mapsto z)$. 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

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

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

$$N \mid \prod_{j\in V'} M_j = N \mid M_k \mid \prod_{j\in V} M_j = M \mid \prod_{j\in V} M_j$$

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 $\prod_{j\in V} M_j$: for $k \in V$, via *OUT* and *PAR*, we have

$$\mathrm{RHS} = M \mid \prod_{j\in V} M_j \xrightarrow{M_k} M \mid \prod_{k\neq j\in V} M_j =: \mathrm{RHS}'$$

Due to *ABORT*, we derive an internal step and release the message M_k

$$\mathrm{LHS} = M \mid \big(\sum_{j\in J} R_j\big)_B^V \xrightarrow{\tau} M \mid \big(\sum_{j\in J} R_j\big)_{B+k}^{V-k} \mid M_k \xrightarrow{M_k} M \mid \big(\sum_{j\in J} R_j\big)_{B+k}^{V-k} =: \mathrm{LHS}'$$

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

A.3.2 Simplifications due to homomorphic encodings/decodings

The remaining proofs in this section have in common that they exhibit particular transitions of terms by constructing appropriate inference trees from either

- the inductive structure of (annotated) terms, or
- inductive inference trees for transitions of other terms which are related to the term under investigation by an inductive encoding $\mathcal{A}, \mathcal{F}$ or decoding $\mathcal{U}_\flat, \mathcal{U}_\sharp$.

Since the $\mathcal{A}, \mathcal{F}, \mathcal{U}_\flat, \mathcal{U}_\sharp$-functions are each defined $[\![\ ]\!]$-homomorphically on every constructor but choice according to the scheme in Section 3, 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 transition inference trees which involve choice rules

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

it suffices for all proofs to regard choice terms in isolation. When looking for simulations between terms and their translations (as in the proofs A.3.3, A.3.4, A.3.5 and A.3.6), 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 A.3.9 and A.3.8) 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.

A.3.3 Proof of Proposition 6.3.4: $\mathcal{F}$ is strong bisimulation

We show the proof for strong synchronous bisimulation, which implies the asynchronous case. The proof is by structural induction on $A \in \mathbb{A}$ and transition induction on $A \rightarrow A'$. By the simplification discussed in Section A.3.2, it suffices to regard the case

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

where, according to the rules in Table 3, there are three subcases. The proof in each case is 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.

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

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

$$
\begin{array}{ll}
\mathcal{F}[\![\, (\sum_{j\in J} R_j)^{V}_{B} \,]\!] & \\
\quad = & (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{b}\rangle \ \mid \prod_{j\in J\setminus(V\cup B)} Init_l\langle\, \mathcal{F}[\![\, R_j \,]\!]\,\rangle \ \mid \ \prod_{j\in V} Lock_l\langle\, \mathcal{F}[\![\, R_j\,]\!]\,\rangle \ \Big) \\
\quad = & (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{b}\rangle \ \mid \prod_{j\in J\setminus(V\cup B)} y_j(x).Lock_l\langle\, \mathcal{F}[\![\, R_j \,]\!]\,\rangle \ \mid \ \prod_{j\in V} Lock_l\langle\, \mathcal{F}[\![\, R_j\,]\!]\,\rangle\sigma_j \ \Big) \\
\quad \xrightarrow{y_k\langle z\rangle} & (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{b}\rangle \ \mid \prod_{k\neq j\in J\setminus(V\cup B)} Init_l\langle\, \mathcal{F}[\![\, R_j \,]\!]\,\rangle \ \mid \ \prod_{j\in V} Lock_l\langle\, \mathcal{F}[\![\, R_j\,]\!]\,\rangle\sigma_j \\
 & \qquad\qquad\qquad\qquad\qquad\qquad \mid \quad Lock_l\langle\, \mathcal{F}[\![\, R_k\,]\!]\,\rangle\{^{z}/_{x}\} \ \Big) \\
\quad = & (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{b}\rangle \ \mid \prod_{j\in J\setminus(V'\cup B)} Init_l\langle\, \mathcal{F}[\![\, R_j \,]\!]\,\rangle \ \mid \ \prod_{j\in V'} Lock_l\langle\, \mathcal{F}[\![\, R_j\,]\!]\,\rangle\sigma'_j \ \Big) \\
\quad = & \mathcal{F}[\![\, (\sum_{j\in J} R_j)^{V'}_{B} \,]\!]
\end{array}
$$

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

case *ABORT* In every context, for $B \neq \emptyset$ and $k \in V$, via rule *ABORT*, w e have

$$
(\sum_{j\in J} R_j)^{V}_{B} \xrightarrow{\tau} (\sum_{j\in J} R_j)^{V-k}_{B+k} \mid M_k \qquad\qquad \text{and}
$$

$$
\begin{array}{l}
\mathcal{F}[\![\, (\sum_{j\in J} R_j)^{V}_{B} \,]\!] \\
\quad = (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{f}\rangle \ \mid \prod_{j\in J\setminus(V\cup B)} Init_l\langle\, \mathcal{F}[\![\, R_j \,]\!]\,\rangle \ \mid \ \prod_{j\in V} Lock_l\langle\, \mathcal{F}[\![\, R_j\,]\!]\,\rangle\sigma_j \ \Big) \\
\quad \xrightarrow{\tau} (\boldsymbol{\nu} l)\Big(\prod_{j\in J\setminus(V\cup B)} Init_l\langle\, \mathcal{F}[\![\, R_j \,]\!]\,\rangle \mid \prod_{k\neq j\in V} Lock_l\langle\, \mathcal{F}[\![\, R_j\,]\!]\,\rangle\sigma_j \mid Exit^{\boldsymbol{f}}_l\langle\, \mathcal{F}[\![\, R_k\,]\!]\,\rangle\sigma_k \Big) \\
\quad \overset{\boldsymbol{f}}{\equiv} (\boldsymbol{\nu} l)\Big(\ \bar{l}\langle \boldsymbol{f}\rangle \ \mid \prod_{j\in J\setminus(V\cup B)} Init_l\langle\, \mathcal{F}[\![\, R_j \,]\!]\,\rangle \ \mid \ \prod_{j\in V-k} Lock_l\langle\, \mathcal{F}[\![\, R_j\,]\!]\,\rangle\sigma_j \ \mid \ \overline{y_k}\langle x\rangle\sigma_k \ \Big) \\
\quad = \mathcal{F}[\![\ (\sum_{j\in J} R_j)^{V-k}_{B+k} \mid M_k \ ]\!]
\end{array}
$$

which holds since $J \setminus (V-k) \setminus (B+k) = J \setminus (V\cup B)$ and $M_k = \overline{y_k}\langle x\rangle\sigma_k$.

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

$$
(\sum_{j\in J} R_j)^{V}_{B} \xrightarrow{\tau} P_k\sigma_k \mid (\sum_{j\in J} R_j)^{V-k}_{B+k} \qquad\qquad \text{and}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathcal{F}[\![\, \big(\sum_{j\in J} R_j\big)^{V}_{B}]\!]} \\
= & (\boldsymbol{\nu} l)\Big(\;\bar{l}\langle \boldsymbol{t}\rangle \;\mid\; \prod_{j\in J\setminus(V\cup B)} \mathit{Init}_l\langle\, \mathcal{F}[\![R_j]\!] \,\rangle \;\mid\; \prod_{j\in V} \mathit{Lock}_l\langle\, \mathcal{F}[\![R_j]\!] \,\rangle\sigma_j \;\Big) \\
\xrightarrow{\tau} & (\boldsymbol{\nu} l)\Big(\prod_{j\in J\setminus(V\cup B)} \mathit{Init}_l\langle\, \mathcal{F}[\![R_j]\!] \,\rangle \mid \prod_{k\neq j\in V} \mathit{Lock}_l\langle\, \mathcal{F}[\![R_j]\!] \,\rangle\sigma_j \mid \mathit{Exit}^{\boldsymbol{t}}_l\langle\, \mathcal{F}[\![R_k]\!] \,\rangle\sigma_k\Big) \\
\overset{\boldsymbol{t}}{\equiv} & (\boldsymbol{\nu} l)\Big(\;\bar{l}\langle \boldsymbol{f}\rangle \mid \prod_{j\in J\setminus(V\cup B)} \mathit{Init}_l\langle\, \mathcal{F}[\![R_j]\!] \,\rangle \mid \prod_{j\in V-k} \mathit{Lock}_l\langle\, \mathcal{F}[\![R_j]\!] \,\rangle\sigma_j \mid \mathcal{F}[\![R_k]\!]\sigma_k \;\Big) \\
\equiv & \mathcal{F}[\![\; R_k\sigma_k \mid \big(\sum_{j\in J} R_j\big)^{V-k}_{B+k} \;]\!]
\end{array}
$$

which holds since $J\setminus(V-k)\setminus(B+k) = J\setminus(V\cup B)$ and $\mathcal{F}[\![R_k\sigma_k]\!] = \mathcal{F}[\![R_k]\!]\sigma_k$. $\square$

A.3.4 Proof of Lemma 6.3.1: $\mathcal{A}$ is weak simulation up to expansion

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 $S \;\rightarrow S'$. By the simplification discussed in Section A.3.2, it suffices to regard the case

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

where, according to the rules in Table 1, there is only one subcase.

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

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

$$
\begin{array}{rcll}
\mathcal{A}[\![\sum_{j\in J} R_j]\!] = \big(\sum_{j\in J} \mathcal{A}[\![R_j]\!]\big)^{\emptyset}_{\emptyset} & \xrightarrow{y_k\langle z\rangle} & \big(\sum_{j\in J} \mathcal{A}[\![R_j]\!]\big)^{(k\mapsto z)}_{\emptyset} & \\
& \xrightarrow{\tau} & \mathcal{A}[\![P_k]\!]\sigma_k \mid \big(\sum_{j\in J} \mathcal{A}[\![R_j]\!]\big)^{\emptyset}_{k} & \\
& \gtrsim & \mathcal{A}[\![P_k]\!]\sigma_k & = \;\mathcal{A}[\![P_k\{^z\!/_x\}]\!]
\end{array}
$$

where the $\gtrsim$ holds due to Lemma 6.2.2. The definition of σ_k concludes the proof. $\square$

A.3.5 Proof of Lemma 6.4.2: $\mathcal{U}_\flat$ is weak simulation

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 A.3.2, it suffices to regard the case

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

where, by the operational rules in Table 3, there are three subcases.

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

Since we know by Lemma 6.2.2 that $\big(\sum_{j\in J} R_j\big)^V_B \gtrsim \prod_{j\in V} M_j$,

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 asynchronous simulation, but we have to care about the simulation in all contexts, including matching messages. Therefore:

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

$$\overline{y_k}\langle z\rangle \mid \big(\sum_{j\in J} R_j\big)^V_B \xrightarrow{\tau} \big(\sum_{j\in J} R_j\big)^{V+(k\mapsto z)}_B \qquad \text{and}$$

$$\mathcal{U}_\flat[\![\, \overline{y_k}\langle z\rangle \mid \big(\sum_{j\in J} R_j\big)^V_B \,]\!] \;=\; \overline{y_k}\langle z\rangle \mid \sum_{j\in J} R_j \mid \prod_{j\in V} M_j \;\equiv\; \mathcal{U}_\flat[\![\, \big(\sum_{j\in J} R_j\big)^{V+(k\mapsto z)}_B \,]\!]$$

immediately concludes the proof by an empty weakly simulating sequence.

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

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

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

$$\big(\sum_{j\in J} R_j\big)^V_\emptyset \xrightarrow{\tau} P_k\sigma_k \mid \big(\sum_{j\in J} R_j\big)^{V-k}_k$$

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

$$\mathcal{U}_\flat[\![\, \big(\sum_{j\in J} R_j\big)^V_\emptyset \,]\!] \;=\; \sum_{j\in J} R_j \mid \prod_{j\in V} M_j \;\xrightarrow{\tau}\; P_k\{{}^{V(k)}\!/\!_z\} \mid \prod_{k\neq j\in V} M_j \;=\; \mathcal{U}_\flat[\![\, P_k\sigma_k \mid \big(\sum_{j\in J} R_j\big)^{V-k}_k \,]\!]$$

which conclude the proof. $\square$

A.3.6 Proof of Lemma 6.4.3: $\mathcal{U}_\sharp^{-1}$ is weak simulation

The proof is by structural induction on $A \in \mathbb{A}$ and transition induction on $\mathcal{U}_\sharp[\![\, A \,]\!] \to S'$. By the simplification discussed in Section A.3.2, it suffices to regard the case

$$A = \big(\sum_{j\in J} R_j\big)^V_B$$

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

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

where, according to the rules in Table 1, there is only one subcase for generating transitions: *C-INP*. For $k \in J$, we have

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

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

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

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

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

case *OUT*: For $k \in V$, the transitions $\mathcal{U}_\sharp[\![A]\!] \xrightarrow{\overline{y}\langle V(k)\rangle} \prod_{j \in V-k} M_j =: S_k$

can be simulated by $A \xrightarrow{\tau} \big(\sum_{j \in J} R_j\big)^{V-k}_{B+k} \mid M_k \xrightarrow{\overline{y}\langle V(k)\rangle} \big(\sum_{j \in J} R_j\big)^{V-k}_{B+k} =: A_k$

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

case (*partial*) $B = \emptyset \neq V$ **:** Let $k = \textit{take}(V)$. Then $\mathcal{U}_\sharp[\![A]\!] = \prod_{j \in V-k} M_j \mid \mathcal{U}_\sharp[\![P_k]\!]\sigma_k =: S'$.

By $A \xrightarrow{\tau} \big(\sum_{j \in J} R_j\big)^{V-k}_{k} \mid P_k\sigma_k =: A'$ such that $S' = \mathcal{U}_\sharp[\![A']\!]$

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. □

A.3.7 Proof of proposition 6.4.4: $\mathcal{U}_\flat$ is progressing.

Proof: 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 subsequent cases where the τ-step may be simulated trivially such that for $n > k_A$:

$$A = A_0 \xrightarrow{\tau} A_1 \xrightarrow{\tau} A_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} A_n \qquad \text{implies} \qquad \mathcal{U}_\flat[\![\, A \,]\!] \xrightarrow{\tau}{}^{+} \mathcal{U}_\flat[\![\, A_n \,]\!].$$

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 6.4.2. 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/E-INP/R-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 in the transition. By inspection of rule *READ* and the definition of the decoding, 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 init- or lock-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 init-state, determined by $|J \setminus (V \cup B)|$, tells how many subsequent applications of *READ* might be possible, before a *COMMIT*-step has to be derived. In a committed choice, the number of its branches in lock-state (provided by $|V|$) yields the number of possible subsequent applications of *ABORT*; afterwards, no more τ-steps can be generated from this choice occurrence, 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 by the sum of all of their branches in either init- or lock-state.

□

A.3.8 Proof of Lemma 6.4.6: $\mathbb{S}$-coupling

The proof is by structural induction on $A \in \mathbb{A}$. By the simplification discussed in Section A.3.2, it suffices to regard the case

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

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

case $($***initial***$)$ $V = \emptyset = B$ **or** $($***committed***$)$ $B \neq \emptyset$ **:** Immediate by Fact 6.4.5.

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

$$\begin{array}{rcl} \mathcal{U}_\flat[\![\,A\,]\!] & = & \prod_{j\in V} M_j \mid \sum_{j\in J} \mathcal{U}_\flat[\![\,R_j\,]\!] \\ & \xrightarrow{\tau} & \prod_{j\in V-k} M_j \mid \mathcal{U}_\flat[\![\,P_k\,]\!]\sigma_k = \prod_{j\in V-k} M_j \mid \mathcal{U}_\sharp[\![\,P_k\,]\!]\sigma_k = \mathcal{U}_\sharp[\![\,A\,]\!]. \end{array}$$

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 $A \Rightarrow A'$. $\square$

A.3.9 Proof of Lemma 6.4.7: Existence of fully committed derivatives

The proof is by structural induction on $A \in \mathbb{A}$. By the simplification discussed in Section A.3.2, it suffices to regard the case

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

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

case $($***initial***$)$ $V = \emptyset = B$ **or** $($***committed***$)$ $B \neq \emptyset$ **:** Immediate with $A' \overset{\text{def}}{=} A$.

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

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

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

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. $\square$

References

[ACS96] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On Bisimulations for the Asynchronous π-Calculus. Technical Report CNRS/INRIA Sophia-Antipolis, submitted for publication, March 1996.

[AH92] S. Arun-Kumar and Matthew Hennessy. An Efficiency Preorder for Processes. *Acta Informatica*, 29:737–760, 1992. Previously published as Computer Science Report 90:05, University of Sussex.

[ALT95] Roberto Amadio, Lone Leth, and Bent Thomsen. From a Concurrent λ-Calculus to the π-Calculus. In Horst Reichel, editor, *Proceedings of 10th International Conference on Fundamentals of Computation Theory (FCT '95, Dresden, Germany)*, volume 965 of *Lecture Notes in Computer Science*, pages 106–115. Springer, 1995. Full version as Technical Report ECRC-95-18.

[Ama93] Roberto Amadio. On the Reduction of CHOCS Bisimulation to π-Calculus Bisimulation. In Best [Bes93], pages 112–126. Extended version as Rapport de Recherche 1786, INRIA-Lorraine, 1993.

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

[Bes93] Eike Best, editor. *Fourth International Conference on Concurrency Theory (CONCUR '93)*, volume 715 of *Lecture Notes in Computer Science*. Springer, 1993.

[BG95] Nadia Busi and Roberto Gorrieri. Distributed Conflicts in Communicating Systems. In Paolo Ciancarini, Oscar Nierstrasz, and Akinori Yonezawa, editors, *Object-Based Models and Languages for Concurrent Systems (Bologna, Italy, July 1994)*, volume 924 of *Lecture Notes in Computer Science*, pages 49–65. Springer, 1995. Also as University of Bologna Technical Report UBLCS-94-8.

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

[But94] Karl-Heinz Buth. Simulation of SOS Definitions with Term Rewriting Systems. In Sannella [San94b], pages 150–164.

[FG96] Cédric Fournet and Georges Gonthier. The Reflexive Chemical Abstract Machine and the Join-Calculus. In POPL '96 [POP96], pages 372–385.

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

[Gla93] Rob van Glabbeek. The Linear Time – Branching Time Spectrum II: The semantics of sequential systems with silent moves (Extended Abstract). In Best [Bes93], pages 66–81.

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

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

[HT91] Kohei Honda and Mario Tokoro. An Object Calculus for Asynchronous Communication. In P. America, editor, *ECOOP '91*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 1991.

[HT92] Kohei Honda and Mario Tokoro. On Asynchronous Communication Semantics. In M. Tokoro, O. Nierstrasz, and P. Wegner, editors, *Object-Based Concurrent Computing 1991*, volume 612 of *Lecture Notes in Computer Science*, pages 21–51. Springer, 1992.

[HY93] Kohei Honda and Nobuko Yoshida. On Reduction-Based Process Semantics. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science*, pages 373–387. Springer, 1993. Full version published in *Theoretical Computer Science* 152(2):437–486, 1995.

[HY94a] Kohei Honda and Nobuko Yoshida. Combinatory Representation of Mobile Processes. In *21st Annual Symposium on Principles of Programming Languages (POPL)*, pages 348–360. ACM Press, January 1994.

[HY94b] Kohei Honda and Nobuko Yoshida. Replication in Concurrent Combinators. In M. Hagiya and J. C. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 786–805. Springer, 1994.

[Jon93] Cliff Jones. A Pi-Calculus Semantics for an Object-Based Design Notation. In Best [Bes93], pages 158–172.

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

[Lav93] Carolina Lavatelli. Non-determinisitc lazy λ-calculus vs. π-calculus. Technical Report LIENS-93-15, Ecole Normale Supérieure, Paris, September 1993.

[Li83] Wei Li. *An Operational Approach to Semantics and Translation for Concurrent Programming Languages.* PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, January 1983. Report CST-20-83.

[Li94] B. Li. A π-calculus specification of Prolog. In Sannella [San94b], pages 379–393.

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

[Mil87] Mark Millington. *Theories of Translation Correctness for Concurrent Porgramming Languages.* PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, August 1987. Report CST-46-87.

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

[Mil90] Robin Milner. Functions as Processes. In M. S. Paterson, editor, *Seventeenth Colloquium on Automata, Languages and Programming (ICALP) (Warwick, England)*, volume 443 of *Lecture Notes in Computer Science*, pages 167–180. Springer, 1990. Previous version as Rapport de Recherche 1154, INRIA Sophia-Antipolis, 1990. Final version published in *Mathematical Structures in Computer Science* 2(2):119–141, 1992.

[Mil91] Robin Milner. The polyadic π-calculus: A tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991. Published in *Logic and Algebra of Specification*, Proceedings of International NATO Summer School 1991, Springer, 1993.

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

[MP95] Ugo Montanari and Marco Pistore. Concurrent Semantics for the π-calculus. In Steve Brookes, Michael Main, Austin Melton, and Michael Mislove, editors, *Proceedings of the Eleventh Annual Conference on Mathematical Foundations of Programming Semantics, (Tulane University, New Orleans, LA, March 29 – April 1, 1995)*, volume 1 of *Electronic Notes in Theoretical Computer Science.* Elesevier Science Publishers, 1995.

[MPW89] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Part I/II. Technical Report ECS-LFCS-89-85/86, Laboratory for Foundations of Computer Science, University of Edinburgh, June 1989. Published in *Information and Computation* 100:1–77, 1992.

[MS92] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Nineteenth Colloquium on Automata, Languages and Programming (ICALP) (Wien, Austria)*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.

[Nie96] Joachim Niehren. Functional Computation as Concurrent Computation. In POPL '96 [POP96].

[Ode95a] Martin Odersky. Applying π: Towards a Basis for Concurrent Imperative Programming. In Uday S. Reddy, editor, *Second ACM SIGPLAN Workshop on State in Programming Languages*, pages 95–108. Department of Computer Science, University of Illinois at Urbana-Champaign (San Francisco, California), January 22 1995. Available as Technical Report UIUCDCS-R-95-1900 or via `http://vesuvius.cs.uiuc.edu:8080/sipl/-index.html`.

[Ode95b] Martin Odersky. Polarized Name Passing. In *Proceedings of 15th Conference on Foundations of Software Technology and Theoretical Computer Science (Bangalore, India, December 18–20, 1995)*, volume 1026 of *Lecture Notes in Computer Science.* Springer, 1995.

[OR94] Vincent van Oostrom and Femke van Raamsdonk. Comparing Combinatory Reduction Systems and Higher-Order Rewrite Systems. In J. Heering, K. Meinke, B. Möller, and T. Nipkow, editors, *Higher-Order Algebra, Logic and Term Rewriting 1993*, volume 816 of *Lecture Notes in Computer Science*, pages 276–304. Springer, 1994.

[Par95] Joachim Parrow. Trios in Concert. (Draft), July 1995.

[Pie96] Benjamin C. Pierce. Programming in the Pi-Calculus: An Experiment in Programming Language Design. Distributed with the Pict implementation, 1996.

[POP96] *23rd Annual Symposium on Principles of Programming Languages (POPL) (St. Petersburg Beach, Florida)*. ACM Press, January 1996.

[Pri78] Lutz 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, Austria).

[PS92] Joachim Parrow and Peter Sjödin. Multiway Synchronization Verified with Coupled Simulation. In Rance Cleaveland, editor, *Third International Conference on Concurrency Theory (CONCUR '92, Stony Brook, NY)*, volume 630 of *Lecture Notes in Computer Science*, pages 518–533. Springer, 1992.

[PS94] Joachim Parrow and Peter Sjödin. The Complete Axiomatization of cs-Congruence. In P. Enjalbert, E. W. Mayr, and K. W. Wagner, editors, *STACS '94*, volume 775 of *Lecture Notes in Computer Science*, pages 557–568. Springer, 1994.

[PT95] Benjamin C. Pierce and David N. Turner. Concurrent Objects in a Process Calculus. In Takayasu Ito and Akinori Yonezawa, editors, *Theory and Practice of Parallel Programming (TPPP, Sendai, Japan, 1994)*, volume 907 of *Lecture Notes in Computer Science*, pages 187–215. Springer, 1995.

[PT96] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. Technical report in preparation; available electronically, 1996.

[Rep92] John Reppy. *Higher-Order Concurrency*. PhD thesis, Cornell University, June 1992. Technical Report TR 92-1285.

[Ros92] Brian Ross. A π-Calculus Semantics of Logical Variables and Unification. In S. Purushothaman and A. Zwarico, editors, *NAPAW '92, Proceedings of the First North American Process Algebra Workshop (Stony Brook, NY)*, pages 13.1–13.14. Johns Hopkins University, Pennsylvania State University, 1992. Available as PennState Technical Report 92-17.

[San92] Davide Sangiorgi. The Lazy Lambda Calculus in a Concurrency Scenario. In *Seventh Annual Symposium on Logic in Computer Science (LICS) (Santa Cruz, California)*, pages 102–109. IEEE, Computer Society Press, June 1992. Earlier version as Report ECS-LFCS-91-189, University of Edinburgh. Final version published in *Information and Computation* 111(1):120–131, 1994.

[San93] Davide Sangiorgi. From π-Calculus to Higher-Order π-Calculus — and Back. In M.-C. Gaudel and J.-P. Jouannaud, editors, *TAPSOFT '93*, volume 668 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 1993.

[San94a] Davide Sangiorgi. An Investigation into Functions as Processes. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics 1993*, volume 802 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 1994.

[San94b] D. Sannella, editor. *Fifth European Symposion on Programming (ESOP '94)*, volume 788 of *Lecture Notes in Computer Science*. Springer, 1994.

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

[San95b] Davide Sangiorgi. On the Bisimulation Proof Method. In J. Wiedemann and P. Hajek, editors, *Twentienth Mathematical Foundations of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, pages 479–488. Springer, 1995. Available as Edinburgh Technical Report ECS-LFCS-94-299.

[San95c] Davide Sangiorgi. π-Calculus, Internal Mobility and Agent-Passing Calculi. Rapport de Recherche RR-2539, INRIA Sophia-Antipolis, 1995. Extracts of parts of the material contained in this paper can be found in the Proceedings of TAPSOFT'95 and ICALP'95.

[San95d] Davide Sangiorgi. Some Thoughts on Asynchrony. Unpublished note, April 1995.

[Smo94] Gert Smolka. A Foundation for Higher-Order Concurrent Constraint Programming. In J.-P. Jouannaud, editor, *Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 50–72. Springer, 1994. Available as Research Report RR-94-16 from DFKI Kaiserslautern.

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

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

[Vaa90] Frits Vaandrager. Process algebra semantics of POOL. In Jos Baeten, editor, *Application of Process Algebra*, pages 173–236. Cambridge University Press, 1990. Earlier version: CWI-Report CS-R8629.

[VP96] Björn Victor and Joachim Parrow. Constraints as processes. submitted for publication, March 1996.

[Wal90] David Walker. Bisimulation and Divergence. *Information and Computation*, 85(2):202–241, 1990. Extended abstract appeared in LICS' 88: 186–192.

[Wal91a] David Walker. π-calculus Semantics of Object-Oriented Programming Languages. In Takayasu Ito and Albert Meyer, editors, *Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pages 532–547. Springer, 1991. Available as Report ECS-LFCS-90-122, University of Edinburgh.

[Wal91b] David Walker. Some Results on the π-Calculus. In A. Yonezawa and T. Ito, editors, *Concurrency: Theory, Languages, and Architecture*, volume 491 of *Lecture Notes in Computer Science*, pages 21–35. Springer, 1991.

[Wal92] David Walker. Objects in the π-calculus. Research Report CS-RR-217, University of Warwick, April 1992. Final version published in *Information and Computation* 116(2):253–271, 1995.

[Wal93] David Walker. Process Calculus and Parallel Object-Oriented Programming Languages. In *International Summer Institute on Parallel Architectures, Languages, and Algorithms,* *Prague*, 1993. Available as Research Report CS-RR-242, Department of Computer Science, University of Warwick.

[Wal94] David Walker. Algebraic Proofs of Properties of Objects. In Sannella [San94b].

Contents