Structural Rewriting in the π-Calculus

David Sabel

Goethe-University, Frankfurt am Main
sabel@ki.cs.uni-frankfurt.de

Abstract

We consider reduction in the synchronous π-calculus with replication, without sums. Usual definitions of reduction in the π-calculus use a closure w.r.t. structural congruence of processes. In this paper we operationalize structural congruence by providing a reduction relation for pi-processes which also performs necessary structural conversions explicitly by rewrite rules. As we show, a subset of structural congruence axioms is sufficient. We show that our rewrite strategy is equivalent to the usual strategy including structural congruence w.r.t. the observation of barbs and thus w.r.t. may- and should-testing equivalence in the pi-calculus.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, F.3.2 Semantics of Programming Languages, D.3.1 Formal Definitions and Theory

Keywords and phrases Process calculi, Rewriting, Semantics

Digital Object Identifier 10.4230/OASIcs.WPTE.2014.51

1 Introduction

The π-calculus [9, 8, 21] is a well-known core model for concurrent and mobile processes with message passing. Its syntax includes parallel process-composition, named channels, and input/output-capabilities on the channels. The data flow in programs of the π-calculus is represented by communication between processes. Since links between processes can be sent as messages, the induced network of processes behaves dynamically.

Even though the π-calculus has been investigated for several decades, its analysis is still an active research topic. One reason is that variants of the π-calculus have a lot of applications even outside the field of computer science. Examples are the Spi-calculus [1] to reason about cryptographic protocols, the stochastic π-calculus [14] with applications in molecular biology, and the use of the π-calculus to model business processes.

The operational behavior of π-processes is defined in terms of a reduction semantics (and often extended by an action semantics, for semantic reasoning), which is built by a single reduction rule for exchanging a message, and applying this reduction rule in so-called reduction contexts. Additionally, a notion of structural congruence is used, which can be applied to processes before and after the reduction step. Structural congruence allows conversions like commuting the order of parallel processes, moving the scope of binders, etc. Unfortunately the complexity of structural congruence is very high. Indeed in its original formulation by Milner, it is even unknown whether structural congruence is decidable. A recent result [22] shows that it is at least EXPSPACE-hard. For reasoning on reductions and semantics of processes the *implicit* use of structural congruence is difficult and also error-prone. This becomes even more difficult, when such proofs are automated. Hence, in this paper we make the conversion w.r.t. structural congruence *explicit* by including the congruence axioms as separate reduction rules in the reduction relation. Moreover, we also simplify the conversion by dropping some rules, which are not necessary for defining the reduction relation (but they may be used as program optimizations).

1st International Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE'14).
Editors: Manfred Schmidt-Schauß, Masahiko Sakai, David Sabel, and Yuki Chiba; pp. 51–62

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our main result is that the semantics of π-processes remains unchanged if our new reduction relation replaces the original one. As semantics we use barbed may- and should-testing equivalence [4] which holds for two processes if their input and output capabilities coincide if the processes are plugged into any program context. This notion of process equivalence is directly based on the reduction semantics and – unlike other process equivalences – like bisimulation or barbed congruence (see e.g. [21]) – it is insensitive for reduction traces. Hence barbed may- and should-testing equivalence can be viewed as the coarsest notion of process equivalence which distinguishes obviously different processes (see [4] for a deep analysis of the different notions of process equivalences and their relation).

Outline In Sect. 2 we recall the synchronous π-calculus and its reduction semantics which includes structural congruence, and we define may- and should-testing equivalence. In Sect. 3 we introduce the simplified reduction semantics – called $\mathcal{D}$-standard reduction – which makes conversions w.r.t. structural congruence explicit. In Sect. 4 we show that may- and should-testing equivalence remains unchanged if the $\mathcal{D}$-standard reduction is used. We conclude in Sect. 5.

2 The Synchronous π-Calculus with May- and Should-Testing

We consider the synchronous π-calculus with replication but without sums (and recursion).

▶ **Definition 2.1** (Syntax of the π-Calculus)**.** Let $\mathcal{N}$ be a countably infinite set of *names*. *Processes* P and *action prefixes* π are defined by the following grammar, where $x, y \in \mathcal{N}$:

$$\begin{array}{lcl} P & ::= & \pi.P \mid P_1 \mid P_2 \mid \mathbf{!}\,P \mid \mathbf{0} \mid \nu x.P \\ \pi & ::= & x(y) \mid \overline{x}\langle y\rangle \end{array}$$

The prefix $x(y)$ is called an *input-prefix*, and $\overline{x}\langle y\rangle$ is called an *output-prefix*.

Names are *bound* by the ν-binder (in $\nu x.P$ name x is bound with scope P) and by the input-prefix (in $x(y).P$ name y is bound with scope P). This induces a notion of α-renaming and α-equivalence as usual. We treat α-equivalent processes as equal. If necessary, we make α-renaming explicit and denote α-equivalence by $=_\alpha$. We use $\mathsf{fn}(P)$ ($\mathsf{fn}(\pi)$, resp.) for the set of *free names* of process P (prefix π, resp.) and $\mathsf{bn}(P)$ ($\mathsf{bn}(\pi)$, resp.) for the set of *bound names* of process P (prefix π, resp.). Note that $\mathsf{fn}(x(y)) = \{x\}$, $\mathsf{fn}(\overline{x}\langle y\rangle) = \{x, y\}$, $\mathsf{bn}(x(y)) = \{y\}$, and $\mathsf{bn}(\overline{x}\langle y\rangle) = \emptyset$. We assume the distinct name convention, i.e. free names are distinct from bound names and bound names are pairwise distinct.

A process $x(y).P$ has the capability to *receive* some name z along the channel named x and then behaves like $P[z/y]$ where $[z/y]$ is the capture free substitution of name y by name z. A process $\overline{x}\langle y\rangle.P$ has the capability to *send* a name y along the channel named x. The *silent process* $\mathbf{0}$ has no capabilities to communicate. $P_1 \mid P_2$ is the *parallel composition* of processes P_1 and P_2. $\nu z.P$ *restricts* the scope of the name z to process P. As a notation we use $\nu\mathcal{X}.P$ abbreviating $\nu x_1.\ldots.\nu x_n.P$. We also use set-notation for $\mathcal{X}$ and e.g. write $x \in \mathcal{X}$ with its obvious meaning. $\mathbf{!}\,P$ is the *replication* of process P, i.e. it can be interpreted as infinitely many copies of P running in parallel.

▶ **Definition 2.2** (Contexts)**.** A *process context* $C \in \mathcal{C}$ is a process with a *hole* $[\cdot]$ at process-position, i.e. $C \in \mathcal{C} ::= [\cdot] \mid \pi.C \mid C \mid P \mid P \mid C \mid \mathbf{!}\,C \mid \nu x.C$. For a context C and a process P the construct $C[P]$ denotes the process where the hole of C is replaced by process P. For contexts C_1, C_2 we say C_1, C_2 *are prefix disjoint* iff there does not exist a context C_3 with

$C_1 = C_2[C_3]$ or $C_2 = C_1[C_3]$. For $P = C[Q]$ the *replication depth* of Q in P w.r.t. C is the number m of replications above Q which is exactly the number of contexts C_i of the form $!\,[\cdot]$ if $C = C_1 \dots C_n$ where C_i are contexts of hole-depth 1.

▶ **Definition 2.3** (Structural congruence $\equiv$)**.** Structural congruence $\equiv$ is the smallest congruence on processes satisfying the equations:

$$\begin{array}{rcl@{\qquad}rcl} P & \equiv & Q, \text{ if } P =_\alpha Q & \nu z.\nu w.P & \equiv & \nu w.\nu z.P \\ P_1 \mid (P_2 \mid P_3) & \equiv & (P_1 \mid P_2) \mid P_3 & \nu z.\mathbf{0} & \equiv & \mathbf{0} \\ P_1 \mid P_2 & \equiv & P_2 \mid P_1 & \nu z.(P_1 \mid P_2) & \equiv & P_1 \mid \nu z.P_2, \text{ if } z \notin \mathsf{fn}(P_1) \\ P \mid \mathbf{0} & \equiv & P & !\,P & \equiv & P \mid !\,P \end{array}$$

It is unknown whether structural congruence of processes is decidable (see also [2, 3, 6] for more discussion and further results), at least it is a very hard problem:

▶ **Proposition 2.4** ([22, Corollary 4.4])**.** *The decision problem whether for two π-processes $P \equiv Q$ holds is EXPSPACE-hard.*

We define the usual reduction semantics of π-processes (see e.g. [21]) as a small-step reduction relation. The only reduction rule is the rule (ia) for communication (interaction):

$$x(y).P \mid \overline{x}\langle v\rangle.Q \xrightarrow{ia} P[v/y] \mid Q$$

Reduction contexts $\mathcal{D}$ are process contexts where the hole is not below a replication or a π-prefix, i.e. $D \in \mathcal{D} ::= [\cdot] \mid D \mid P \mid P \mid D \mid \nu x.D$ where $x \in \mathcal{N}$ and where P is a process.

▶ **Definition 2.5** (Standard Reduction, $\xrightarrow{sr}$)**.** With $\xrightarrow{\mathcal{D},ia}$ we denote the closure of $\xrightarrow{ia}$ w.r.t. reduction contexts, and a *standard reduction* $\xrightarrow{sr}$ consists of applying a $\xrightarrow{\mathcal{D},ia}$-reduction modulo structural congruence, i.e.

$$\frac{P \xrightarrow{ia} Q}{D[P] \xrightarrow{\mathcal{D},ia} D[Q]} \text{ where } D \in \mathcal{D} \qquad\qquad \frac{P \equiv P' \ \wedge \ P' \xrightarrow{\mathcal{D},ia} Q' \ \wedge \ Q' \equiv Q}{P \xrightarrow{sr} Q}$$

We use the following notation for unions of transformations, where a transformation is some binary relation on processes (e.g. $\xrightarrow{sr}$):

▶ **Definition 2.6.** For a transformation $\xrightarrow{a}$ we define $\xrightarrow{a,0} := \{(P,P) \mid P \text{ is a process}\}$ and $\xrightarrow{a,i} := \{(P,Q) \mid P \xrightarrow{a} S \wedge S \xrightarrow{a,i-1} Q\}$ for $i > 0$. Transitive and reflexive-transitive closure are defined as $\xrightarrow{a,+} := \bigcup_{i>0} \xrightarrow{a,i}$, and $\xrightarrow{a,*} := \bigcup_{i\geq 0} \xrightarrow{a,i}$. For $\xrightarrow{a}$ and $\xrightarrow{b}$ let $\xrightarrow{a\vee b} := \xrightarrow{a} \cup \xrightarrow{b}$.

▶ **Example 2.7.** An example for a sequence of standard reductions is

$$\nu x.(\overline{x}\langle w\rangle.\mathbf{0} \mid x(y).\overline{z}\langle y\rangle.\mathbf{0}) \mid !\,z(u).\mathbf{0} \xrightarrow{sr} \nu x.(\overline{z}\langle w\rangle.\mathbf{0}) \mid !\,z(u).\mathbf{0} \xrightarrow{sr} !\,z(u).\mathbf{0}$$

Making the conversions w.r.t. $\equiv$ more explicit we can write this as:

$$\begin{array}{rl} & \nu x.(\overline{x}\langle w\rangle.\mathbf{0} \mid x(y).\overline{z}\langle y\rangle.\mathbf{0}) \mid !\,z(u).\mathbf{0} \equiv \nu x.(x(y).\overline{z}\langle y\rangle.\mathbf{0} \mid \overline{x}\langle w\rangle.\mathbf{0}) \mid !\,z(u).\mathbf{0} \\ \xrightarrow{\mathcal{D},ia} & \nu x.(\overline{z}\langle w\rangle.\mathbf{0} \mid \mathbf{0}) \mid !\,z(u).\mathbf{0} \equiv \nu x.(\overline{z}\langle w\rangle.\mathbf{0}) \mid !\,z(u).\mathbf{0} \equiv \nu x.(\overline{z}\langle w\rangle.\mathbf{0} \mid !\,z(u).\mathbf{0}) \\ \equiv & \nu x.(\overline{z}\langle w\rangle.\mathbf{0} \mid (z(u).\mathbf{0} \mid !\,z(u).\mathbf{0})) \equiv \nu x.((z(u).\mathbf{0} \mid \overline{z}\langle w\rangle.\mathbf{0}) \mid !\,z(u).\mathbf{0}) \\ \xrightarrow{\mathcal{D},ia} & \nu x.((\mathbf{0} \mid \mathbf{0}) \mid !\,z(u).\mathbf{0}) \equiv \nu x.(\mathbf{0} \mid !\,z(u).\mathbf{0}) \equiv \nu x.\mathbf{0} \mid !\,z(u).\mathbf{0} \equiv \mathbf{0} \mid !\,z(u).\mathbf{0} \equiv !\,z(u).\mathbf{0} \end{array}$$

The high complexity of deciding structural congruence justifies making structural conversion visible during reduction. I.e., instead of using $\equiv$ implicitly during reduction, it makes sense to introduce the conversions as separate reduction rules. Moreover, not all axioms of structural congruence are required to apply $\xrightarrow{\mathcal{D},ia}$-steps, e.g. the axiom $\mathbf{0} \mid P \equiv P$ is not necessary, and also it is not necessary to apply the axioms below a replication or a π-prefix. Before defining a modified reduction for the π-calculus (in Sect. 3) we define the semantics of π-processes by a process equivalence. In the π-calculus several definitions and notions for process equivalences exist. We choose the approach of testing the input and output capabilities of processes in all contexts and also test whether the capability may or should occur. This notion of may- and should-testing (sometimes also called must- or fair must testing, see e.g. [11, 7, 4, 16]) is close to contextual equivalence [10, 13] in other program calculi like the lambda calculus, but adapted to the concurrent setting (see e.g. [12, 17, 18, 19, 23] for contextual equivalence with may- and should-semantics in extended concurrent lambda calculi). A strong connection between may- and should-testing equivalence and a classic notion of contextual equivalence (using a notion of success) for the π-calculus was shown in [20]. May- and should-testing equivalence is a coarse notion of program equivalence equating as much processes as possible, but discriminating obviously different processes.

Input and output capabilities are formalized by the notion of a barb:

▶ **Definition 2.8** (Barb)**.** A process P *has a barb on input* x (written as $P\!\Rsh^{x}$) iff P can receive a name on channel x, i.e. $P = \nu\mathcal{X}.(x(y).P' \mid P'')$ where $x \notin \mathcal{X}$, and P *has a barb on output* x (written as $P\!\Rsh^{\overline{x}}$) iff P can emit a name on channel x, i.e. $P = \nu\mathcal{X}.(\overline{x}\langle y\rangle.P' \mid P'')$ where $x \notin \mathcal{X}$. We write $P \equiv\!\Rsh^{x}$ ($P \equiv\!\Rsh^{\overline{x}}$, resp.) iff $P \equiv P'$ and $P'\!\Rsh^{x}$ ($P'\!\Rsh^{\overline{x}}$, resp.).

As observations for process equivalence we will use on the one hand whether a process may reduce to a process that has a barb, and on the other hand whether a process has the ability to have a barb on every reduction path:

▶ **Definition 2.9** (May-barb and Should-barb)**.** For $\mu \in \{x, \overline{x}\}$, P *may have a barb on* μ (written as $P{\downarrow_\mu}$) iff $P \xrightarrow{sr,*} Q \wedge Q \equiv\!\Rsh^{\mu}$, and P *should have a barb on* μ (written as $P{\Downarrow_\mu}$) iff $P \xrightarrow{sr,*} P' \implies P'{\downarrow_\mu}$. We write $P{\uparrow_\mu}$ iff $P{\Downarrow_\mu}$ does not hold, and $P{\Uparrow_\mu}$ iff $P{\downarrow_\mu}$ does not hold.

Note that $P{\uparrow_\mu}$ equivalently means that P can reduce to a process that has no input (or output, resp.) capabilities on the channel μ, i.e. $P{\uparrow_\mu}$ holds iff $P \xrightarrow{sr,*} P'$ and $P'{\Uparrow_\mu}$.

▶ **Definition 2.10** (Barbed May- and Should-Testing Equivalence)**.** Processes P and Q are *barbed testing equivalent* (written $P \sim Q$) iff $P \precsim Q \wedge Q \precsim P$, where $\precsim := \precsim_{\text{may}} \cap \precsim_{\text{should}}$ and

$$P \precsim_{\text{may}} Q \quad \text{iff } \forall x \in \mathcal{N},\ \mu \in \{x, \overline{x}\}, C \in \mathcal{C}\colon\ C[P]{\downarrow_\mu} \implies C[Q]{\downarrow_\mu}$$
$$P \precsim_{\text{should}} Q \text{ iff } \forall x \in \mathcal{N},\ \mu \in \{x, \overline{x}\}, C \in \mathcal{C}\colon\ C[P]{\Downarrow_\mu} \implies C[Q]{\Downarrow_\mu}$$

Our definition of barbed testing equivalence is given in a general form, since the may- and should-behavior, all channel names, the input and output barbs, and also all channels are separately considered. However, in the π-calculus the definition is equivalent to simpler definitions. I.e., it is sufficient to observe the should-behavior only, and to observe input (or output) channels exclusively, and to either observe a single channel name, or existentially observing the barb capabilities (see [4] for the asynchronous π-calculus and [20] for the synchronous variant used in this paper). However, since our results also apply for the general definition, we refrain from working with the simpler definition.

As an easy result, we show that structural congruence preserves the semantics of processes:

▶ **Proposition 2.11.** *If* $P \equiv Q$ *then* $P \sim Q$.

Proof. Let $P \equiv Q$, C be a context s.t. $P{\downarrow_\mu}$ ($P{\uparrow_\mu}$, resp.), i.e. $C[P] \xrightarrow{sr,*} P'$ and $P' \equiv\!\!\Uparrow^\mu$ (or $P' \Uparrow_\mu$, resp.). Since $\equiv$ is a congruence and included in $\xrightarrow{sr}$, we have $C[Q] \equiv C[P] \xrightarrow{sr,*} P'$ and $C[Q] \xrightarrow{sr,*} P'$ which shows $C[Q]{\downarrow_\mu}$ ($C[Q]{\uparrow_\mu}$, resp.). Also $C[Q]{\downarrow_\mu} \implies C[P]{\downarrow_\mu}$ and $C[Q]{\uparrow_\mu} \implies C[P]{\uparrow_\mu}$ hold by symmetry of $\equiv$. Since $S{\uparrow_\mu} \iff \neg S{\Downarrow_\mu}$, this implies $P \sim Q$. ◀

3 Structural Congruence as Rewriting

In this section we make the conversion w.r.t. structural equivalence explicit and also restrict this conversion to reduction contexts. In the following definition the relation $\xrightarrow{sca}$ is the reduction relation corresponding to structural congruence axioms applied in both directions. However, the rewrite rules are a little bit more general than the congruence axioms, but not more general than the structural congruence relation. The relation $\xrightarrow{sc}$ is a restriction of $\xrightarrow{sca}$, the replication axiom is only permitted in the expanding direction, and rules adding, removing and moving down ν-binders as well as adding or removing the silent process are not included. The removed rules can be seen as optimizations, i.e. removing "garbage", but – as we show – they are dispensable for reasoning about may- and should-testing equivalence.

▶ **Definition 3.1** (Structural Reduction)**.** The relation $\xrightarrow{sc}$ is defined by the following rules:

$$
\begin{array}{llll}
(assocl) & P_1 \mid (P_2 \mid P_3) \xrightarrow{sc} (P_1 \mid P_2) \mid P_3 & (replunfold) & !\,P \xrightarrow{sc} P \mid !\,P \\
(assocr) & (P_1 \mid P_2) \mid P_3 \xrightarrow{sc} P_1 \mid (P_2 \mid P_3) & (nuup) & D[\nu z.P] \xrightarrow{sc} \nu z.D[P], \text{ if } z \notin \mathsf{fn}(D), \\
(commute) & P_1 \mid P_2 \xrightarrow{sc} P_2 \mid P_1 & & [\cdot] \neq D \in \mathcal{D}
\end{array}
$$

The relation $\xrightarrow{sca}$ is defined by the rules:

$$
\begin{array}{llll}
 & P \xrightarrow{sca} Q \text{ if } P \xrightarrow{sc} Q & (replfold) & P \mid !\,P \xrightarrow{sca} !\,P \\
(nuintro) & P \xrightarrow{sca} \nu z.P \text{ if } z \notin \mathsf{fn}(P) & (intro0l) & P \xrightarrow{sca} \mathbf{0} \mid P \\
(nurem) & \nu z.P \xrightarrow{sca} P \text{ if } z \notin \mathsf{fn}(P) & (intro0r) & P \xrightarrow{sca} P \mid \mathbf{0} \\
(nudown) & \nu z.D[P] \xrightarrow{sca} D[\nu z.P], \text{ if } z \notin \mathsf{fn}(D), [\cdot] \neq D \in \mathcal{D} & (rem0r) & P \mid \mathbf{0} \xrightarrow{sca} P
\end{array}
$$

The relations $\xrightarrow{\mathcal{D},sc}$ and $\xrightarrow{\mathcal{C},sca}$ are defined as:

$$
\frac{P \xrightarrow{sc} Q}{D[P] \xrightarrow{\mathcal{D},sc} D[Q]} \text{ where } D \in \mathcal{D} \qquad \frac{P \xrightarrow{sca} Q}{C[P] \xrightarrow{\mathcal{C},sca} C[Q]} \text{ where } C \in \mathcal{C}
$$

We sometimes add more information on the reduction arrow, and e.g. write $\xrightarrow{\mathcal{D},sc,nuup,*}$ for a (maybe empty) finite sequence of $\xrightarrow{\mathcal{D},sc}$-transformations which all apply the rule $(nuup)$.

▶ **Lemma 3.2.** *The relation* $\xrightarrow{\mathcal{C},sca,*}$ *coincides with structural congruence, i.e.* $\xrightarrow{\mathcal{C},sca,*} = \equiv$.

Since $\xrightarrow{\mathcal{C},sca,*}$ and $\equiv$ coincide, we can replace $\equiv$ in the definition of standard reduction:

▶ **Corollary 3.3.** $P \xrightarrow{sr} Q$ *iff* $P \xrightarrow{\mathcal{C},sca,*} P' \;\wedge\; P' \xrightarrow{\mathcal{D},ia} Q' \;\wedge\; Q' \xrightarrow{\mathcal{C},sca,*} Q$.

We define our new variant of standard reduction, called $\mathcal{D}$-standard reduction, which restricts the conversion w.r.t. structural congruence to $\xrightarrow{\mathcal{D},sc}$-transformations:

▶ **Definition 3.4** ($\mathcal{D}$-Standard Reduction, $\xrightarrow{dsr}$)**.** A *$\mathcal{D}$-standard reduction* $\xrightarrow{dsr}$ applies the rule (ia) in a reduction context $D \in \mathcal{D}$ modulo $\xrightarrow{\mathcal{D},sc,*}$, i.e. :

$$
\frac{P \xrightarrow{\mathcal{D},sc,*} P' \;\wedge\; P' \xrightarrow{\mathcal{D},ia} Q' \;\wedge\; Q' \xrightarrow{\mathcal{D},sc,*} Q}{P \xrightarrow{dsr} Q}
$$

▶ **Example 3.5.** We consider the same process as in Example 2.7.

$$\begin{aligned} & \nu x.(\overline{x}\langle w\rangle.\mathbf{0} \mid x(y).\overline{z}\langle y\rangle.\mathbf{0}) \mid !\, z(u).\mathbf{0} \xrightarrow{\mathcal{D},sc} \nu x.(x(y).\overline{z}\langle y\rangle.\mathbf{0} \mid \overline{x}\langle w\rangle.\mathbf{0}) \mid !\, z(u).\mathbf{0} \\ \xrightarrow{\mathcal{D},ia} \; & \nu x.(\overline{z}\langle w\rangle.\mathbf{0} \mid \mathbf{0}) \mid !\, z(u).\mathbf{0} \xrightarrow{\mathcal{D},sc,5} \nu x.(\mathbf{0} \mid ((\overline{z}\langle w\rangle.\mathbf{0} \mid z(u).\mathbf{0}) \mid !\, z(u).\mathbf{0})) \\ \xrightarrow{\mathcal{D},ia} \; & \nu x.(\mathbf{0} \mid ((\mathbf{0} \mid \mathbf{0}) \mid !\, z(u).\mathbf{0})) \end{aligned}$$

Note that $\mathcal{D}$-standard reduction cannot remove the $\mathbf{0}$-components and the ν-binder.

Replacing standard reduction by $\mathcal{D}$-standard reduction results in modified observation predicates and a modified definition of may- and should-testing equivalence. However, we will show that the modified equivalence coincides with the original one in Theorem 4.13.

▶ **Definition 3.6.** For $\mu \in \{x, \overline{x}\}$, P *may have a barb w.r.t.* $\xrightarrow{dsr}$ *on* x (written as $P{\downarrow_{\mathcal{D},\mu}}$) iff $P \xrightarrow{dsr,*} Q \xrightarrow{\mathcal{D},sc,*} Q'$ and $Q' \not\mapsto^{\mu}$, and P *should have a barb w.r.t.* $\xrightarrow{dsr}$ *on* x (written as $P{\Downarrow_{\mathcal{D},\mu}}$) iff for all processes P' such that $P \xrightarrow{dsr,*} P'$, also $P'{\downarrow_{\mathcal{D},\mu}}$ holds. We write $P{\uparrow_{\mathcal{D},\mu}}$ iff $\neg(P{\Downarrow_{\mathcal{D},\mu}})$ holds and $P \Uparrow_{\mathcal{D},\mu}$ iff $\neg(P \downarrow_{\mathcal{D},\mu})$ holds. Processes P and Q are *barbed testing equivalent w.r.t.* $\xrightarrow{dsr}$ (written $P \sim_{\mathcal{D}} Q$) iff $P \precsim_{\mathcal{D}} Q \wedge Q \precsim_{\mathcal{D}} P$, where $\precsim_{\mathcal{D}} := \precsim_{\mathcal{D},\text{may}} \cap \precsim_{\mathcal{D},\text{should}}$ and

$$\begin{aligned} P \precsim_{\mathcal{D},\text{may}} Q \quad & \text{iff } \forall x \in \mathcal{N},\ \mu \in \{x, \overline{x}, C \in \mathcal{C}\}\colon\ C[P]{\downarrow_{\mathcal{D},\mu}} \implies C[Q]{\downarrow_{\mathcal{D},\mu}} \\ P \precsim_{\mathcal{D},\text{should}} Q & \text{ iff } \forall x \in \mathcal{N},\ \mu \in \{x, \overline{x}\}, C \in \mathcal{C}\}\colon\ C[P]{\Downarrow_{\mathcal{D},\mu}} \implies C[Q]{\Downarrow_{\mathcal{D},\mu}} \end{aligned}$$

Note that $P{\uparrow_{\mathcal{D},\mu}}$ is equivalent to: $\exists Q : P \xrightarrow{dsr,*} Q \wedge Q \Uparrow_{\mathcal{D},\mu}$. Our main result will be that $\sim$ and $\sim_{\mathcal{D}}$ coincide and thus $\mathcal{D}$-standard reduction can be used for reasoning about process equivalence. For showing $\sim = \sim_{\mathcal{D}}$ it is sufficient to prove that ${\downarrow_{\mu}} = {\downarrow_{\mathcal{D},\mu}}$ and ${\Downarrow_{\mu}} = {\Downarrow_{\mathcal{D},\mu}}$.

4 Relating Reduction Strategies

As a required notation we define conversions w.r.t. $\equiv$ that are not included in $\xrightarrow{\mathcal{D},sc}$:

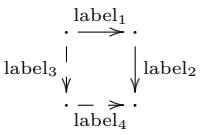
▶ **Definition 4.1** (Internal $\xrightarrow{\mathcal{C},sca}$-Transformations, $\xrightarrow{isca}$)**.** With $\xrightarrow{isca}$ we denote a $\xrightarrow{\mathcal{C},sca}$ transformation that is not a $\xrightarrow{\mathcal{D},sc}$ transformation, i.e. $\xrightarrow{isca} = \xrightarrow{\mathcal{C},sca} \setminus \xrightarrow{\mathcal{D},sc}$. With $\xrightarrow{isca\langle k\rangle}$ we denote a $\xrightarrow{isca}$-transformation at replication depth k, i.e. k is the number of replications above the redex of the $\xrightarrow{isca}$-transformation.

In the remainder of this section we establish our main result by showing that $\sim$ and $\sim_{\mathcal{D}}$ coincide. The proof is structured into three parts: in Sect. 4.1 we show that for a sequence of standard reductions $P \xrightarrow{sr,*} Q$ the internal conversions $\xrightarrow{isca}$ can be shifted to the right resulting in a reduction $P \xrightarrow{dsr,*} Q' \xrightarrow{isca,*} Q$. In Sect. 4.2 we show that if a process has a barb w.r.t. $\equiv$, i.e. $P \equiv \not\mapsto^{\mu}$, then we can remove internal conversions $\xrightarrow{isca}$ and thus $P \xrightarrow{\mathcal{D},sc,*} P'$ s.t. $P' \not\mapsto^{\mu}$. In Sect. 4.3 we use the results of both previous sections to show ${\downarrow_{\mu}} = {\downarrow_{\mathcal{D},\mu}}$ and ${\Downarrow_{\mu}} = {\Downarrow_{\mathcal{D},\mu}}$ which implies the coincidence of $\sim$ and $\sim_{\mathcal{D}}$.

4.1 Shifting Internal Conversions to the End

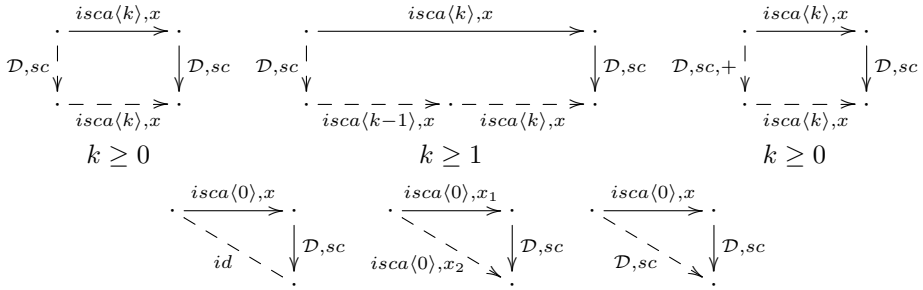
In this section we show that for every standard reduction sequence $P \xrightarrow{sr,*} Q$, also a $\mathcal{D}$-standard reduction sequence followed by internal structural conversion steps exists, i.e. $P \xrightarrow{dsr,*} Q' \xrightarrow{isca,*} Q$. The result is established by inspecting overlappings of the forms $P_1 \xrightarrow{isca} P_2 \xrightarrow{\mathcal{D},sc} P_3$ and $P_1 \xrightarrow{isca} P_2 \xrightarrow{\mathcal{D},ia} P_3$, where in both cases the $\xrightarrow{isca}$-transformation must be commuted with the other reduction or transformation.

We will use so-called commuting diagrams as a notation, which are diagrams of the form as shown on the right. Every arrow represents a transformation step or a sequence of steps denoted by $*$ (0 or more steps) or $+$ (1 or more steps) as part of the label. Labels denote the used reduction rule, solid arrows mean given transformations, dashed arrows are existentially quantified transformations. If the label is *id*, then this means that processes are identical. Labels may include variables x, x_i for the name of a used rule. These variables are meant universally quantified for the diagram, i.e. all occurrences of x in one diagram can only be instantiated with the same rule name x, and for diagrams with different variables (e.g. x_1 and x_2) any variable can be instantiated with different (or equal) rule names. If not otherwise stated x, x_1, x_2 may be instantiated with (*assocl*), (*assocr*), (*commute*), (*nucomm*), (*nudownr*), (*nuup*), (*replunfold*), (*replfold*), (*nuintro*), (*intro0l*), (*intro0r*), (*rem0r*), or (*nurem*).



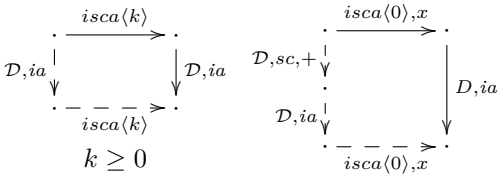
Inspecting all overlappings between a $\xrightarrow{\mathcal{D},sc}$- and an $\xrightarrow{isca}$-step shows:

▶ **Lemma 4.2.** *Given a sequence $P_1 \xrightarrow{isca} P_2 \xrightarrow{\mathcal{D},sc} P_3$, then the sequence can always be commuted by one of the following diagrams:*



Inspecting all overlappings between a $\xrightarrow{\mathcal{D},ia}$- and an $\xrightarrow{isca}$-step

▶ **Lemma 4.3.** *Given a sequence $P_1 \xrightarrow{isca} P_2 \xrightarrow{\mathcal{D},ia} P_3$, then the sequence can always be commuted by one of the following diagrams:*



Using the diagrams of the two previous lemmas we are able to show that in a $\xrightarrow{sr}$-reduction sequence all $\xrightarrow{isca}$-steps can always be shifted to the right end.

▶ **Proposition 4.4.** *Let $P_1 \xrightarrow{a_1} P_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} P_n$ where every $\xrightarrow{a_i}$ is either an $\xrightarrow{\mathcal{C},sca}$ or a $\xrightarrow{\mathcal{D},ia}$ transformation. Then there exists $P_1 \xrightarrow{b_1} Q_1 \xrightarrow{b_2} \ldots \xrightarrow{b_m} Q_m \xrightarrow{isca,*} P_n$ where every $\xrightarrow{b_i}$ is either a $\xrightarrow{\mathcal{D},ia}$ or a $\xrightarrow{\mathcal{D},sc}$ transformation.*

Proof. In the input sequence (and also intermediate sequences in the proof) we can switch to the representation where $\xrightarrow{a_i}$ is either an $\xrightarrow{isca\langle k\rangle}$ (for some $k \geq 0$) or a $\xrightarrow{\mathcal{D},ia}$, or a $\xrightarrow{\mathcal{D},sc}$ transformation. We represent the two latter cases uniformly by $\xrightarrow{\mathcal{D},sc\vee ia}$ and thus only have to deal with sequences consisting of $\xrightarrow{isca\langle k\rangle}$- and $\xrightarrow{\mathcal{D},sc\vee ia}$-steps. The diagrams of Lemmas 4.2 and 4.3 can be seen as rewriting rules on such sequences, where we also simplify

the representation resulting in the rules:

$$\xrightarrow{isca\langle k\rangle} . \xrightarrow{\mathcal{D},sc\vee ia} \rightsquigarrow \xrightarrow{\mathcal{D},sc\vee ia} . \xrightarrow{isca\langle k-1\rangle} . \xrightarrow{isca\langle k\rangle} \text{ for } k \geq 1 \quad (1)$$

$$\xrightarrow{isca\langle k\rangle} . \xrightarrow{\mathcal{D},sc\vee ia} \rightsquigarrow \xrightarrow{\mathcal{D},sc\vee ia,n} . \xrightarrow{isca\langle k\rangle} \text{ for } k \geq 0 \text{ and any } n \geq 1 \quad (2)$$

$$\xrightarrow{isca\langle 0\rangle} . \xrightarrow{\mathcal{D},sc\vee ia} \rightsquigarrow \varepsilon \text{ (where } \varepsilon \text{ represents the empty string)} \quad (3)$$

$$\xrightarrow{isca\langle 0\rangle} . \xrightarrow{\mathcal{D},sc\vee ia} \rightsquigarrow \xrightarrow{isca\langle 0\rangle} \quad (4)$$

$$\xrightarrow{isca\langle 0\rangle} . \xrightarrow{\mathcal{D},sc\vee ia} \rightsquigarrow \xrightarrow{\mathcal{D},sc\vee ia} \quad (5)$$

Let $\mathfrak{R} = P_1 \xrightarrow{a_1} P_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} P_n$. We define a measure $\mu(\mathfrak{R})$ for those reduction sequences: let $\mathsf{P}(\mathfrak{R})$ be the longest prefix of $\mathfrak{R}$ that has an $\xrightarrow{\mathcal{D},sc\vee ia}$-step as its last step, or the empty sequence, if there is no $\xrightarrow{\mathcal{D},sc\vee ia}$-step in $\mathfrak{R}$. W.l.o.g. $\mathsf{P}(\mathfrak{R}) = P_1 \xrightarrow{a_1} P_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{j-1}} P_j$. Now the multiset $\mu(\mathfrak{R})$ is constructed by inserting the pair $\#(P_i \xrightarrow{isca\langle k\rangle} P_{i+1})$ for every reduction step $P_i \xrightarrow{isca\langle k\rangle} P_{i+1}$ in $\mathsf{P}(\mathfrak{R})$ (and inserting no pairs for steps $P_i \xrightarrow{\mathcal{D},sc\vee ia} P_{i+1}$), where

$$\#(P_i \xrightarrow{isca\langle k\rangle} P_{i+1}) = \begin{cases} (k,\infty), \text{ if } \xrightarrow{isca\langle k\rangle} \text{ is not the last } \xrightarrow{isca} \text{ reduction in } \mathsf{P}(\mathfrak{R}) \\ (k,d), \quad \text{otherwise, where } d = j-i-1 \text{ (i.e. } d \text{ is the number of} \\ \qquad\quad \text{reductions in } \mathsf{P}(\mathfrak{R}) \text{ that follow after } P_i \xrightarrow{isca\langle k\rangle} P_{i+1}) \end{cases}$$

We use the multiset ordering, where pairs are ordered lexicographically. The ordering is well-founded and if the multiset $\mu(\mathfrak{R})$ is empty, then $\mathsf{P}(\mathfrak{R})$ does not contain $\xrightarrow{isca}$-transformations. We show the claim by induction on $\mu(\mathfrak{R})$: If $\mu(\mathfrak{R})$ is empty then $\mathsf{P}(\mathfrak{R})$ only contains $\xrightarrow{\mathcal{D},sc\vee ia}$-steps (or is empty). Hence $\mathfrak{R}$ is of the right form and the claim holds. For the induction step assume that $\mu(\mathfrak{R})$ contains at least one pair, hence $\mathsf{P}(\mathfrak{R})$ has at least a subsequence $\xrightarrow{isca}\xrightarrow{\mathcal{D},sc\vee ia}$. Let $\mathsf{P}(\mathfrak{R}) = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{isca\langle k\rangle} P_{i+1} \xrightarrow{\mathcal{D},sc\vee ia} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m}$ and $\mathfrak{R} = \mathsf{P}(\mathfrak{R}) \xrightarrow{isca,*} P_n$. We apply one of the rewriting rules derived from the diagrams to the subsequence $P_i \xrightarrow{isca\langle k\rangle} P_{i+1} \xrightarrow{\mathcal{D},sc\vee ia} P_{i+2}$, Let $\mathfrak{R}'$ be the sequence $\mathfrak{R}$ after applying the rewrite rule. We inspect the cases for all five rules (1)–(5):

(1) $\mathfrak{R}' = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{\mathcal{D},sc\vee ia} P' \xrightarrow{isca\langle k-1\rangle} P'' \xrightarrow{isca\langle k\rangle} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m} \xrightarrow{isca,*} P_n$. For $\mu(\mathfrak{R}')$ compared to $\mu(\mathfrak{R})$ there are two possibilities:

- If $m = 0$, then $\mathsf{P}(\mathfrak{R}') = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{\mathcal{D},sc\vee ia} P'$ and a pair $(k,1)$ is removed and perhaps some other pair (k,∞) is replaced by (k,d) for some $d < \infty$.
- If $m > 0$, then $\mathsf{P}(\mathfrak{R}') = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{\mathcal{D},sc\vee ia} P' \xrightarrow{isca\langle k-1\rangle} P'' \xrightarrow{isca\langle k\rangle} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m}$ and a pair $(k,m+1)$ is replaced by two pairs (k,m) and $(k-1,\infty)$.

In both cases $\mu(\mathfrak{R}') < \mu(\mathfrak{R})$ and thus the induction hypothesis can be applied to $\mathfrak{R}'$.

(2) $\mathfrak{R}' = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{\mathcal{D},sc\vee ia,m'} P' \xrightarrow{isca\langle k\rangle} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m} \xrightarrow{isca,*} P_n$ where $m' \geq 1$. For $\mu(\mathfrak{R}')$ compared to $\mu(\mathfrak{R})$ there are two possibilities:

- If $m = 0$, then $\mathsf{P}(\mathfrak{R}') = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{\mathcal{D},sc\vee ia,m'} P'$ and a pair $(k,1)$ is removed and perhaps some other pair (k,∞) is replaced by (k,d) where $d < \infty$.
- If $m > 0$, then $\mathsf{P}(\mathfrak{R}') = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{\mathcal{D},sc\vee ia,m'} P' \xrightarrow{isca\langle k\rangle} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m}$ and a pair $(k,m+1)$ is replaced by (k,m).

In both cases $\mu(\mathfrak{R}') < \mu(\mathfrak{R})$ and thus the induction hypothesis can be applied to $\mathfrak{R}'$.

(3) $\mathfrak{R}' = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i = P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m} \xrightarrow{isca,*} P_n$. For $\mu(\mathfrak{R}')$ compared to $\mu(\mathfrak{R})$ there are two possibilities:

- If $m = 0$, then $\mathsf{P}(\mathfrak{R}')$ is a prefix of $P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i$ and a pair $(k, 1)$ is removed and perhaps a pair (k, ∞) is replaced by (k, d) where $d < \infty$.
- If $m > 0$, then $\mathsf{P}(\mathfrak{R}') = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i = P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m}$ and a pair $(k, m+1)$ is removed, and perhaps a pair (k, ∞) is replaced by (k, d) for some $d < \infty$.

In both cases $\mu(\mathfrak{R}') < \mu(\mathfrak{R})$ and thus the induction hypothesis can be applied to $\mathfrak{R}'$.

(4) $\mathfrak{R}' = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{isca\langle k\rangle} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m} \xrightarrow{isca,*} P_n$. For $\mu(\mathfrak{R}')$ compared to $\mu(\mathfrak{R})$ there are two possibilities:

- If $m = 0$, then $\mathsf{P}(\mathfrak{R}')$ is a prefix of $P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i$ and a pair $(k, 1)$ is removed and perhaps some other pair (k, ∞) is replaced by (k, d) where $d < \infty$.
- If $m > 0$, then $\mathsf{P}(\mathfrak{R}') = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{isca\langle k\rangle} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m}$ and a pair $(k, m+1)$ is replaced by (k, m).

(5) $\mathfrak{R}' = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{\mathcal{D},sc\vee ia} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m} \xrightarrow{isca,*} P_n$. For $\mu(\mathfrak{R}')$ compared to $\mu(\mathfrak{R})$ there are two possibilities:

- $\mathsf{P}(\mathfrak{R}')$ is empty, since all $\xrightarrow{a_j}$-steps are $\xrightarrow{\mathcal{D},sc\vee ia}$-steps. Then $\mathfrak{R}'$ is of the right form.
- $\mathsf{P}(\mathfrak{R}') = P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_i} P_i \xrightarrow{\mathcal{D},sc\vee ia} P_{i+2} \xrightarrow{\mathcal{D},sc\vee ia,m} P_{i+2+m}$. Then a pair (k, m) is removed and another pair (k, ∞) is replaced by a pair (k, d) where $d < \infty$ (and $d > m$). In this case $\mu(\mathfrak{R}') < \mu(\mathfrak{R})$ and thus we can apply the induction hypothesis to $\mathfrak{R}'$. ◀

▶ **Remark 4.5.** Termination of the rewriting in the previous proof can also be shown automatically by encoding the rules as an (extended) term rewriting system (TRS) and then using a termination prover. Let K,N, and X be variables, S, Z be constructor symbols encoding Peano-numbers, and *isca*, *dscdia*, and *gen* be defined function symbols. The encoding of the rules (1)–(5) is straight-forward, except for rule (2) due to the constraint $n \geq 1$. It is encoded by three TRS-rules (2), (2'), (2"), where the first rule "guesses" the number n, and the rules (2') and (2") generate the corresponding number of $\xrightarrow{\mathcal{D},sc\vee ia}$-steps:

$$\begin{aligned}
isca(S(k), dscdia(\mathsf{X})) &\to dscdia(isca(\mathsf{K}, isca(S(\mathsf{K}), \mathsf{X}))) && (1)\\
isca(\mathsf{K}, dscdia(\mathsf{X})) &\to gen(S(\mathsf{N}), isca(\mathsf{K}, \mathsf{X})) && (2)\\
gen(S(\mathsf{N}), \mathsf{X}) &\to dscdia(gen(\mathsf{N}, \mathsf{X})) && (2')\\
gen(Z, \mathsf{X}) &\to \mathsf{X} && (2'')\\
isca(Z, dscdia(\mathsf{X})) &\to \mathsf{X} && (3)\\
isca(Z, dscdia(\mathsf{X})) &\to isca(Z, \mathsf{X}) && (4)\\
isca(Z, dscdia(\mathsf{X})) &\to dscdia(Z, \mathsf{X}) && (5)
\end{aligned}$$

Thus the TRS has rules with free variables on the right hand side. However, the termination prover AProVE [5] has shown innermost-termination of the TRS (which is sufficient), and the certifier CeTA [24] has certified the termination proof[1]. A similar encoding approach was already used for automating correctness proofs for program transformations in [15].

We conclude this subsection by showing that also the number of (ia)-reductions is preserved by shifting internal transformations:

▶ **Theorem 4.6.** ***1.*** *If $P \xrightarrow{dsr,n} Q$ then $P \xrightarrow{sr,n} Q$.* ***2.*** *If $P \xrightarrow{sr,n} Q$ then $P \xrightarrow{dsr,n} Q' \xrightarrow{isca,*} Q$.*

Proof. Part (1) holds, since every $\xrightarrow{dsr}$-step is also an $\xrightarrow{sr}$-step. For part (2), let $P = P_0 \xrightarrow{sr} P_1 \ldots \xrightarrow{sr} P_n = Q$. Corollary 3.3 shows that for $\leq i \leq n-1$ $P_i \xrightarrow{sr} P_{i+1}$ can be written as $P_i \xrightarrow{\mathcal{C},sca,*} P_i' \xrightarrow{\mathcal{D},ia} P_i'' \xrightarrow{\mathcal{C},sca,*} P_{i+1}$ for some processes P_i', P_i'' and thus $P = P_0 \xrightarrow{\mathcal{C},sca,*} P_0' \xrightarrow{\mathcal{D},ia} P_0'' \xrightarrow{\mathcal{C},sca,*} P_1 \ldots P_{n-1} \xrightarrow{\mathcal{C},sca,*} P_{n-1}' \xrightarrow{\mathcal{D},ia} P_{n-1}'' \xrightarrow{\mathcal{C},sca,*} P_n = Q$. Proposition 4.4 shows that there exists a process Q' s.t. $P \xrightarrow{\mathcal{D},ia\vee sc,*} Q' \xrightarrow{isca,*} Q$. The

[1] the termination proof is available at http://www.ki.cs.uni-frankfurt.de/persons/sabel/picalc.html

construction in the proof of Proposition 4.4 together with the diagrams in Lemmas 4.2 and 4.3 imply that no $\xrightarrow{\mathcal{D},ia}$ transformation is eliminated or introduced. Thus $P \xrightarrow{\mathcal{D},ia\vee sc,*} Q'$ must contain exactly n $\xrightarrow{\mathcal{D},ia}$-transformations and thus it is also a sequence $P \xrightarrow{dsr,n} Q'$. ◀

4.2 Removing Internal Conversions from Barbs

In this section we show that $P \equiv \Gamma^{\mu}$ also implies that $P \xrightarrow{\mathcal{D},sc,*} P'$ s.t. $P' \Gamma^{\mu}$. Let $\mathcal{F}$-contexts be the class of contexts that do not have a hole below an input- or output prefix, i.e.

$$F \in \mathcal{F} ::= [\cdot] \mid !\,F \mid F \mid P \mid P \mid F \mid \nu x.F \quad \text{where } x \in \mathcal{N} \text{ and } P \text{ is a process}$$

We say that $F \in \mathcal{F}$ *captures* the name $x \in \mathcal{N}$ iff the hole of F is in scope of a restriction νx.

▶ **Lemma 4.7.** *If $P \equiv \nu\mathcal{X}.(\pi.Q \mid R)$, then there exists $F \in \mathcal{F}$, a prefix π', and process Q', s.t. $P = F[\pi'.Q']$ and $\mathsf{fn}(\pi) \cap \mathcal{X} = \mathsf{fn}(\pi') \cap \mathsf{fn}(P)$.*

Proof. Let $P_0 \xrightarrow{\mathcal{C},sca,n} P_n = \nu\mathcal{X}.(\pi.Q \mid R)$. We use induction on n. If $n = 0$ then the claim holds, since $\nu\mathcal{X}.([\cdot] \mid R)$ is an $\mathcal{F}$-context. If $n > 0$ let $P_0 \xrightarrow{\mathcal{C},sca} P_1 \xrightarrow{\mathcal{C},sca,n-1} \nu\mathcal{X}.(\pi.Q \mid R)$. The induction hypothesis shows that $P_1 = F_1[\pi_1.Q_1]$ s.t. $\mathsf{fn}(\pi) \cap \mathcal{X} = \mathsf{fn}(\pi_1) \cap \mathsf{fn}(P_1)$. Inspecting all possibilities for the step $P_0 \xrightarrow{\mathcal{C},sca} P_1$, shows that $P_0 = F_0[\pi_0.Q_0]$ for some $\mathcal{F}$-context F_0, and where π_0 is π_1 but perhaps with a renaming of variables due to α-renaming. However, $\xrightarrow{\mathcal{C},sca}$-transformation can neither capture free names nor move bound names out of their scope, and thus the claim of the lemma holds. ◀

▶ **Lemma 4.8.** *If $P = F[\pi.Q]$ for some $F \in \mathcal{F}$, prefix π, and process Q, then $P \xrightarrow{\mathcal{D},sc,*} \nu\mathcal{X}.(\pi'.Q' \mid R)$ s.t. $\mathsf{fn}(\pi) \cap \mathsf{fn}(F[\pi.Q]) = \mathsf{fn}(\pi') \cap \mathcal{X}$.*

Proof. We use structural induction on F. If F is empty, then P is of the required form.

- If $F = F' \mid R$ then by the induction hypothesis (and since $[\cdot] \mid R$ is a $\mathcal{D}$-context): $F'[\pi.Q] \mid R \xrightarrow{\mathcal{D},sc,*} (\nu\mathcal{X}.(\pi'.Q' \mid R')) \mid R$ s.t. $\mathsf{fn}(\pi) \cap \mathsf{fn}(F[\pi.Q]) = \mathsf{fn}(\pi') \cap \mathcal{X}$. Suppose $\mathcal{X} = \{x_1, \ldots, x_m\}$ and let $\mathcal{Y} := \{y_1, \ldots, y_m\}$ be fresh names, and $[\vec{y}/\vec{x}]$ be the substitution $[y_1/x_1, \ldots, y_m/x_m]$. We can extend this reduction as follows: $(\nu\mathcal{X}.(\pi'.Q' \mid R')) \mid R$ $=_\alpha (\nu\mathcal{Y}.(\pi'[\vec{y}/\vec{x}].Q'[\vec{y}/\vec{x}] \mid R'[\vec{y}/\vec{x}])) \mid R \xrightarrow{\mathcal{D},sc,nuup,*} (\nu\mathcal{Y}.((\pi'[\vec{y}/\vec{x}].Q'[\vec{y}/\vec{x}] \mid R'[\vec{y}/\vec{x}]) \mid R))$ $\xrightarrow{\mathcal{D},sc,assocr} (\nu\mathcal{Y}.(\pi'[\vec{y}/\vec{x}].Q'[\vec{y}/\vec{x}] \mid (R'[\vec{y}/\vec{x}] \mid R)))$. Clearly, only bound names of π' are renamed and thus the condition on the names holds.
- If $F = R \mid F'$ then the reasoning is analogous to the previous case.
- If $F = \nu x.(F')$ then by the induction hypothesis $(F'[\pi.Q]) \xrightarrow{\mathcal{D},sc,*} (\nu\mathcal{X}.(\pi'.Q' \mid R'))$ where $\mathsf{fn}(\pi) \cap \mathsf{fn}(F'[\pi.Q]) = \mathsf{fn}(\pi') \cap \mathcal{X}$. The same reduction can be performed in the $\mathcal{D}$-context $\nu x.([\cdot])$: $\nu x.(F'[\pi.Q]) \xrightarrow{\mathcal{D},sc,*} \nu x.(\nu\mathcal{X}.(\pi'.Q' \mid R'))$ and occurrences of x in π and π' are captured for both processes.
- If $F = !\,F'$ then by the induction hypothesis $F'[\pi.Q] \xrightarrow{\mathcal{D},sc,*} (\nu\mathcal{X}.(\pi'.Q' \mid R'))$ where $\mathsf{fn}(\pi) \cap \mathsf{fn}(F'[\pi.Q]) = \mathsf{fn}(\pi') \cap \mathcal{X}$. Suppose that $\mathcal{X} = \{x_1, \ldots, x_m\}$ and let $\mathcal{Y} = \{y_1, \ldots, y_m\}$ be fresh names, $[\vec{y}/\vec{x}]$ be the substitution $[y_1/x_1, \ldots, y_m/x_m]$, and P_0 be an α-renamed copy of $F'[\pi.Q]$. Then $!\,F'[\pi.Q] \xrightarrow{\mathcal{D},sc,replunfold} F'[\pi.Q] \mid !\,P_0 \xrightarrow{\mathcal{D},sc,*} (\nu\mathcal{X}.(\pi'.Q' \mid R')) \mid !\,P_0$ $=_\alpha (\nu\mathcal{Y}.(\pi'[\vec{y}/\vec{x}].Q'[\vec{y}/\vec{x}] \mid R'[\vec{y}/\vec{x}])) \mid !\,P_0 \xrightarrow{\mathcal{D},sc,nuup,*} \nu\mathcal{Y}.((\pi'[\vec{y}/\vec{x}].Q'[\vec{y}/\vec{x}] \mid R'[\vec{y}/\vec{x}]) \mid !\,P_0)$ $\xrightarrow{\mathcal{D},sc,assocr} \nu\mathcal{Y}.(\pi'[\vec{y}/\vec{x}].Q'[\vec{y}/\vec{x}] \mid (R'[\vec{y}/\vec{x}] \mid !\,P_0))$. The last process is of the required form and free names of π' remain free in $\pi'[\vec{y}/\vec{x}]$. ◀

Combining Lemmas 4.7 and 4.8 results in:

▶ **Theorem 4.9.** *If* $P \equiv \nu\mathcal{X}.(\pi.Q \mid R)$ *then* $P \xrightarrow{\mathcal{D},sc,*} \nu\mathcal{X}'.(\pi'.Q' \mid R')$ *s.t.* $\mathsf{fn}(\pi) \cap \mathcal{X} = \mathsf{fn}(\pi') \cap \mathcal{X}'$.

▶ **Corollary 4.10.** *For all* $x \in \mathcal{N}$ *and* $\mu \in \{x, \overline{x}\}$*:* $P \equiv\!\!\Rsh^{\mu}$ *iff* $P \xrightarrow{\mathcal{D},sc,*} Q$ *and* $Q \Rsh^{\mu}$.

Proof. This follows since $\xrightarrow{\mathcal{D},sc,*} \subset \equiv$, and from Theorem 4.9. ◀

4.3 Coincidence of $\sim$ and $\sim_{\mathcal{D}}$

We now prove our main result, by first showing that the observation predicates $\downarrow_{\mu}$,$\Downarrow_{\mu}$ remain unchanged if $\xrightarrow{sr}$ is replaced by $\xrightarrow{dsr}$. This implies that $\sim = \sim_{\mathcal{D}}$.

▶ **Proposition 4.11.** $\downarrow_{\mu} = \downarrow_{\mathcal{D},\mu}$.

Proof. We have to show two parts:

- If $P\downarrow_{\mathcal{D},\mu}$, then $P \xrightarrow{dsr,n} Q \xrightarrow{\mathcal{D},sc,*} Q'$ and $Q' \Rsh^{\mu}$. Theorem 4.6 shows that $P \xrightarrow{sr,n} Q$ and Corollary 4.10 shows that $Q \equiv\!\!\Rsh^{\mu}$. Thus we have $P\downarrow_{\mu}$.
- If $P\downarrow_{\mu}$ then $P \xrightarrow{sr,n} Q \wedge Q \equiv\!\!\Rsh^{\mu}$. Theorem 4.6 shows that $P \xrightarrow{dsr,n} Q' \xrightarrow{isca,*} Q$, Lemma 3.2 implies that $Q' \equiv\!\!\Rsh^{\mu}$, and by Corollary 4.10 $Q' \xrightarrow{dsr,*} Q''$ s.t. $Q'' \Rsh^{\mu}$. This shows $P\downarrow_{\mathcal{D},\mu}$. ◀

▶ **Proposition 4.12.** $\Downarrow_{\mu} = \Downarrow_{\mathcal{D},\mu}$.

Proof. We show the converse, i.e. $\uparrow_{\mu} = \uparrow_{\mathcal{D},\mu}$. Proposition 4.11 already implies $\Uparrow_{\mu} = \Uparrow_{\mathcal{D},\mu}$.

- Let P be a process with $P\uparrow_{\mathcal{D},\mu}$. Then $P \xrightarrow{dsr,n} Q$ and $Q\Uparrow_{\mathcal{D},\mu}$. Theorem 4.6 shows that $P \xrightarrow{sr,n} Q$. Since $Q\Uparrow_{\mathcal{D},\mu} \iff Q\Uparrow_{\mu}$, this implies $P\uparrow_{\mu}$.
- If $P\uparrow_{\mu}$, then $P \xrightarrow{sr,n} Q$ and $Q\Uparrow_{\mu}$. Theorem 4.6 shows that $P \xrightarrow{dsr,n} Q' \xrightarrow{isca,*} Q$. This also implies $Q' \equiv Q$ and thus $Q'\Uparrow_{\mu}$. Since $Q'\Uparrow_{\mathcal{D},\mu} \iff Q'\Uparrow_{\mu}$, this shows $P\uparrow_{\mathcal{D},\mu}$. ◀

The definitions of $\sim$ and $\sim_{\mathcal{D}}$ only differ in the used observation predicates. In the two previous propositions we have shown, that the observation predicates are identical, and thus:

▶ **Theorem 4.13.** $\sim = \sim_{\mathcal{D}}$.

A consequence of the previous theorem and Proposition 2.11 is:

▶ **Corollary 4.14.** *If* $P \equiv Q$ *then* $P \sim_{\mathcal{D}} Q$.

5 Conclusion

We have defined a reduction strategy for the synchronous π-calculus which makes conversions w.r.t. structural congruence explicit by reduction rules, and uses a restricted set of those conversions. We have shown that the new reduction strategy does not change the semantics of processes w.r.t. may- and should-testing equivalence. For further research, we may use the new reduction strategy for proving correctness of process transformations, e.g. automated computation of overlappings between transformation steps and $\mathcal{D}$-standard reductions. Also extensions of the π-calculus, may be the topic of further research.

Acknowledgment. I thank Manfred Schmidt-Schauß for discussions and helpful comments on the paper. I thank René Thiemann for his great support on using AproVE and CeTA. I thank the anonymous reviewers for their valuable comments on the paper.

References

1 M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *CCS'97*, pages 36–47. ACM, 1997.
2 J. Engelfriet and T. Gelsema. A new natural structural congruence in the pi-calculus with replication. *Acta Inf.*, 40(6-7):385–430, 2004.
3 J. Engelfriet and T. Gelsema. An exercise in structural congruence. *Inf. Process. Lett.*, 101(1):1–5, 2007.
4 C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. *J. Log. Algebr. Program.*, 63(1):131–173, 2005.
5 J. Giesl, P. Schneider-Kamp, and R. Thiemann. Automatic termination proofs in the dependency pair framework. In *IJCAR'06*, volume 4130 of *LNCS*, pages 281–286. Springer, 2006.
6 V. Khomenko and R. Meyer. Checking pi-calculus structural congruence is graph isomorphism complete. In *ACSD'09*, pages 70–79. IEEE, 2009.
7 C. Laneve. On testing equivalence: May and must testing in the join-calculus. Technical Report UBLCS 96-04, University of Bologna, 1996.
8 R. Milner. *Communicating and Mobile Systems: the π-calculus.* CUP, 1999.
9 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i & ii. *Inform. and Comput.*, 100(1):1–77, 1992.
10 J. H. Morris. *Lambda-Calculus Models of Programming Languages.* PhD thesis, MIT, 1968.
11 V. Natarajan and R. Cleaveland. Divergence and fair testing. In *ICALP'95*, volume 944 of *LNCS*, pages 648–659. Springer, 1995.
12 J. Niehren, D. Sabel, M. Schmidt-Schauß, and J. Schwinghammer. Observational semantics for a concurrent lambda calculus with reference cells and futures. *Electron. Notes Theor. Comput. Sci.*, 173:313–337, 2007.
13 G. D. Plotkin. Call-by-name, call-by-value, and the lambda-calculus. *Theoret. Comput. Sci.*, 1:125–159, 1975.
14 C. Priami. Stochastic pi-calculus. *Comput. J.*, 38(7):578–589, 1995.
15 C. Rau, D. Sabel, and M. Schmidt-Schauß. Correctness of program transformations as a termination problem. In *IJCAR'12*, volume 7364 of *LNCS*, pages 462–476. Springer, 2012.
16 A. Rensink and W. Vogler. Fair testing. *Inform. and Comput.*, 205(2):125–198, 2007.
17 D. Sabel and M. Schmidt-Schauß. A call-by-need lambda-calculus with locally bottom-avoiding choice: Context lemma and correctness of transformations. *Math. Structures Comput. Sci.*, 18(03):501–553, 2008.
18 D. Sabel and M. Schmidt-Schauß. A contextual semantics for Concurrent Haskell with futures. In *PPDP'11*, pages 101–112. ACM, 2011.
19 D. Sabel and M. Schmidt-Schauß. Conservative concurrency in Haskell. In *LICS'12*, pages 561–570. IEEE, 2012.
20 D. Sabel and M. Schmidt-Schauß. Contextual equivalence for the pi-calculus that can stop. Frank report 53, J. W. Goethe-Universität Frankfurt am Main, 2014. `http://www.ki.cs.uni-frankfurt.de/papers/frank/pi-stop-frank.pdf`.
21 D. Sangiorgi and D. Walker. *The π-calculus: a theory of mobile processes.* CUP, 2001.
22 M. Schmidt-Schauß, C. Rau, and D. Sabel. Algorithms for Extended Alpha-Equivalence and Complexity. In *RTA'13*, volume 21 of *LIPIcs*, pages 255–270. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2013.
23 M. Schmidt-Schauß and D. Sabel. Correctness of an STM Haskell implementation. In *ICFP'13*, pages 161–172. ACM, 2013.
24 R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *TPHOLs'09*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009.