TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Analysing Input/Output Capabilities of Mobile Processes with a Generic Type System

Barbara König

TUM-I0009
SFB-Bericht Nr. 342/05/00 A
April 00

TUM–INFO–04-I0009-0/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Analysing Input/Output-Capabilities of Mobile Processes with a Generic Type System*

Barbara König (koenigb@in.tum.de)

Fakultät für Informatik, Technische Universität München

Abstract. We introduce a generic type system (based on Milner's sort system) for the synchronous polyadic π-calculus, allowing us to mechanise the analysis of input/output capabilities of mobile processes. The parameter of the generic type system is a lattice-ordered monoid, the elements of which are used to describe the capabilities of channels with respect to their input/output-capabilities. The type system can be instantiated in order to check process properties such as upper and lower bounds on the number of active channels, confluence and absence of blocked processes.

1 Introduction

For the analysis and verification of processes there are basically two approaches: methods that are complete, but cannot be fully mechanised, and fully automatic methods which are consequently not complete, i.e. not all processes satisfying the property to be checked are recognised.

One promising direction for the latter approach is to use type or sort systems and type inference with rather complex types abstracting from process behaviour. In the last few years there have been several papers presenting such type systems for the polyadic π-calculus and other process calculi, checking e.g. input/output behaviour [15], absence of deadlocks [7], security properties [1, 4], allocation of permissions to names [16] and many others. Types are compositional and thus allow reuse of information obtained in the analysis of smaller subsystems.

One drawback of the type systems mentioned above is the fact that they are specialised to check very specific properties. A much more general approach is a theory of types by Honda [6] which is based on typed algebras and gives a classification of type systems. This theory is very general and it is thus necessary to prove the subject reduction property and the correctness of a type system for every instance. Our paper attempts to fill the gap between the two extremes. We present a generic type system where we can show the subject reduction property for the general case, and by instantiating the type system we are able to analyse specific properties of processes. Despite its generality, our type system can be used to generate existing type systems, or at least subsets of them. With the introduction of residuation (explained below) we can even type some processes which are not typable by comparable type systems.

We concentrate on properties connected to input/output capabilities of processes in the synchronous polyadic π-calculus. Compared to many other type systems a type of a process in our setting is not a constraint on the environment or context of the process (as

* Research supported by SFB 342 (subproject A3) of the DFG.

in [15]), but rather a representation of the capabilities of the process itself. Composing two processes in our setting means adding their capabilities.

In our examples (see section 5) we check properties such as upper and lower bounds on the number of active channels, confluence, absence of blocked input or output prefixes. Determining these capabilities of a process involves counting and we attempt to keep this concept as general as possible by basing the generic type system on commutative monoids. Instantiating a type system mainly involves choosing an appropriate monoid, and monoid elements associated with input and output prefixes (e.g. for counting the number of prefixes with a certain subject).

Instead of giving the precise answer to every question, our type system uses over-approximation (e.g. we can expect results of the form "there are at most two active channels with subject x at any given time"). Hence plain monoids are not sufficient, but we need ordered monoids (so-called lattice-ordered monoids or l-monoids), equipped with a partial order compatible with summation.

There is a huge class of lattice-ordered monoids which are residuated, i.e. some limited form of subtraction can be defined. Residuation can be put to good use in process analysis. Consider, e.g. the process $P = \bar{x}.x.\mathbf{0}$. While P increases the number of occurrences of the output prefix $\bar{x}$ by one, it does not do so for the input prefix x, since we are interested exclusively in the number of prefixes on the outer level (i.e. in prefixes which are currently active) and x can only be reached by a communication with $\bar{x}$ which decreases the number of input prefixes in the environment by one. This decrease can be anticipated when typing P, and is taken into consideration by subtracting one from the number of input prefixes.

The type of a process contains an assignment of names to sorts and a mapping of sorts to strings of sorts (as in [13]), keeping track of channel arities, i.e. if channel x has sort s, and n-ary tuples are communicated via x, then s will be mapped to a string of sorts having length n, being the sorts of the respective channels. Thus, successful typing also guarantees the absence of runtime errors produced by mismatching arities. Furthermore a monoid element is assigned to each sort s. The monoid element is expected to be an upper bound for the capabilities of all channels having sort s.

The rest of this paper is structured as follows: in section 2 we introduce some preliminaries, by giving a short summary of the π-calculus, presenting lattice-ordered commutative monoids and defining the notion of type (and some operations on types). In section 3 we present the type system and show the subject reduction property for its most general version, i.e. for arbitrary l-monoids. We then demonstrate how the type system can be used for process analysis (section 4), by giving the connection between the type of a process and its input/output capabilities. This section concludes with the sketch of an algorithm for type inference. We will then discuss some examples, related work and directions for future work.

2 Preliminaries

2.1 The π-Calculus

The π-calculus [12, 13] is an influential paradigm describing communication and mobility of processes. In this paper we will consider the synchronous polyadic π-calculus without choice and matching, and replication is only defined for input prefixes. Its syntax is as follows:

$$P ::= \mathbf{0} \mid (\nu x\colon s)P \mid P_1|P_2 \mid \bar{x}\langle\tilde{z}\rangle.P \mid x(\tilde{y}).P \mid !x(\tilde{y}).P$$

where s is an element from a fixed set of sorts S and x is taken from a fixed set of names $\mathcal{N}$. $\tilde{y} = y_1 \dots y_n$ and $\tilde{z} = z_1 \dots z_n$ are abbreviations for sequences with elements from $\mathcal{N}$. We call $\bar{x}\langle\tilde{z}\rangle$ *output prefix* and $x(\tilde{y})$ *input prefix*.

The set of all free names (i.e. names not bound by either ν or by an input prefix) of a process P is denoted by $\mathit{fn}(P)$. The process obtained by replacing the free names y_i by x_i in P (and avoiding capture) is called $P\{\tilde{x}/\tilde{y}\}$.

Structural congruence is the smallest congruence obeying the rules in the upper part of table 1, and equating processes that can be converted into one another by consistent renaming of bound names (α-conversion). We use a reduction semantics as for the chemical abstract machine [2] instead of a labelled transition semantics.

Structural Congruence	(C-COM) $P_1|P_2 \equiv P_2|P_1$ (C-$\mathbf{0}$) $P|\mathbf{0} \equiv P$
	(C-ASS) $P_1|(P_2|P_3) \equiv (P_1|P_2)|P_3$
	(C-RESTR1) $(\nu x\colon s)(\nu y\colon t)P \equiv (\nu y\colon t)(\nu x\colon s)P$ if $x \neq y$
	(C-RESTR2) $((\nu x\colon s)P_1)|P_2 \equiv (\nu x\colon s)(P_1|P_2)$ if $x \notin \mathit{fn}(P_2)$
Reduction Rules	(R-COMM) $\bar{x}\langle\tilde{z}\rangle.Q \mid x(\tilde{y}).P \to Q \mid P\{\tilde{z}/\tilde{y}\}$
	(R-REP) $\bar{x}\langle\tilde{z}\rangle.Q \mid !x(\tilde{y}).P \to Q \mid P\{\tilde{z}/\tilde{y}\} \mid !x(\tilde{y}).P$
	(R-PAR) $\dfrac{P \to P'}{P|Q \to P'|Q}$ (R-RESTR) $\dfrac{P \to P'}{(\nu x\colon s)P \to (\nu x\colon s)P'}$
	(R-EQU) $\dfrac{Q \equiv P, P \to P', P' \equiv Q'}{Q \to Q'}$

Table 1. operational semantics of the π-calculus

Consider the following processes which we will use as an example in this paper (we omit the final $\mathbf{0}$):

$$F = c(r).\bar{d}\langle r\rangle.d(a).\bar{c}\langle a\rangle \quad S = d(s).s(h_1, h_2).\bar{d}\langle h_1\rangle \quad T = \bar{c}\langle h\rangle.c(x) \quad H = \bar{h}\langle i_1, i_2\rangle$$

There is a forwarder F which receives requests on a channel c, forwards them on a channel d to a server, receives the answer and sends it back on c. The server S receives requests on d, and we assume that these requests come with a name s where the server can get further information. The server obtains this information, processes it and sends the answer back on d (in our example we keep the "processing part" very simple, the server just sends back the first component). Furthermore T is a trigger process, starting the execution of F and receiving the result in the end, and H delivers information to the server.

We can combine the processes F, S, T, H to obtain P as the entire system. If we want F and S to be persistent, we regard P'.

$$P = T \mid H \mid (\nu d\colon s_d)(F \mid S) \qquad P' = T \mid H \mid (\nu d\colon s_d)(!F \mid !S)$$

A programmer analysing this piece of code might be interested in the following properties: input/output behaviour, upper and lower bound on the number of channels being active, confluence properties and absence of blocked prefixes that never find a communication partner. E.g., examining P will reveal that at any given time every name is used for input and output at most once and that P is therefore confluent.

2.2 Residuated Lattice-ordered Monoids

Lattice-ordered monoids are a well-developed mathematical concept (see e.g. [3]). We are interested in commutative residuated l-monoids in order to represent input/output capabilities.

Definition 1. (Lattice-ordered Monoid)
A commutative lattice-ordered monoid (l-monoid) *is a tuple* $(I, +, \leq)$ *where* I *is a set,* $+ : I \times I \to I$ *is a binary operation and* $\leq$ *is a partial order which satisfy:*

- $(I, +)$ *is a commutative monoid, i.e.* $+$ *is associative and commutative, and there is a unit* 0 *with* $0 + a = a$ *for every monoid element* $a \in I$*.*
- $(I, \leq)$ *is a lattice, i.e.* $\leq$ *is a partial order, where two elements* $a, b \in I$ *have a* join *(or least upper bound)* $a \vee b$ *and a* meet *(or greatest lower bound)* $a \wedge b$*.*
- I *contains a bottom element* $\bot$*, the smallest element in* I*, and a top element* $\top$*, the greatest element in* I*.*
- *For* $a, b, c \in I$*:* $a + (b \vee c) = (a + b) \vee (a + c)$ *and* $a + (b \wedge c) = (a + b) \wedge (a + c)$

Any l-monoid $(I, +, \leq)$ is associated with an l-monoid $(I, \oplus, \leq)$ where $a \oplus b = (a + b) \vee a \vee b$ and $\bot$ is the unit. The significance of $\oplus$ can be made clear with the following consideration: monoid elements will be used to label sorts, being an upper bound for the capabilities of channels having this sort. E.g., we assume that a free name x and a bound name y have sort s, indicating that, during reduction, x might replace y. The capabilities of x and y are a respectively b. What capability should be associated with s? In the presence of positive monoid elements only, $a + b$ is the correct answer. If, however, a is negative, $a + b$ is actually smaller than b and if x has not yet replaced y, the monoid element associated with s underestimates the capabilities of y. Since we use over-approximation the correct sort label is $a \oplus b$.

Definition 2. (Residuated l-monoid) *Let* $(I, +, \leq)$ *be an l-monoid and let* $a, b \in I$*. The* residual $a - b$ *is the smallest* x *(if it exists) such that* $a \leq x + b$*.* I *is called* residuated *if all residuals* $a - b$ *exist in* I *for* $a, b \in I$*.*

Example: one l-monoid which we will later use for the analysis of processes is $IO = (\{none, I, O, both\}, \vee, \leq)$ where $none \leq I \leq both$, $none \leq O \leq both$ and the monoid operation is the join, i.e. the l-monoid degenerates to a lattice. A channel name has for example capability *O* if it is used at most for output and capability *both* if it may be used for both output and input.

In order to count the number of inputs or outputs we use the l-monoid $\mathbb{Z}^{\infty} = (\mathbb{Z} \cup \{\infty, -\infty\}, +, \leq)$ with all integers including ∞ and $-\infty$ ($\infty + (-\infty) = -\infty$). Residuation is subtraction for all monoid elements different from ∞ and $-\infty$.

The cartesian product of two l-monoids, e.g. $\mathbb{Z}^{\infty} \times \mathbb{Z}^{\infty}$, is also an l-monoid.

We use the following inequations concerning residuated l-monoids: for all elements a, b, c of a residuated l-monoid it holds that

$$a \leq (a - b) + b \qquad (a + b) - b \leq a \qquad (a + b) - c \leq (a - c) + b$$
$$(a + b) \vee 0 \leq (a \vee 0) + (b \vee 0) \qquad a + b \leq a \oplus b \qquad \bot + \bot = \bot \qquad \top + \top = \top$$

And we define: $sig(a) = \begin{cases} \bot \text{ if } a < 0 & 0 \qquad\qquad \text{if } a = 0 \\ \top \text{ if } a > 0 & \textit{undefined} \text{ otherwise} \end{cases}$

3 The Type System and its Properties

We define the notion of types and type assignments which have already been informally introduced in section 1.

Definition 3. (Type Assignment) *Let S be a fixed set of sorts and let $(I, +, \leq)$ be a fixed l-monoid. A type assignment $\Gamma = ob_\Gamma; m_\Gamma; x_1\colon \langle s_1/a_1\rangle, \ldots, x_n\colon \langle s_n/a_n\rangle$ (abbreviated by $ob_\Gamma; m_\Gamma; \tilde{x}\colon \langle \tilde{s}/\tilde{a}\rangle$) consists of a* sort mapping $ob_\Gamma : S \to S^*$ *(mapping sorts to* object sorts*), a mapping $m_\Gamma : S \to I$ (assigning a monoid element to every sort) and an assignment of channel names x_i to tuples consisting of a sort s_i and a monoid element a_i.*

We define $sort_\Gamma(x_i) = s_i$ and $\Gamma, y\colon \langle t/b\rangle$ denotes $ob_\Gamma; m_\Gamma; \tilde{x}\colon \langle \tilde{s}/\tilde{a}\rangle, y\colon \langle t/b\rangle$.

Sorts are used to control the mobility of names. That is if $ob_\Gamma(s) = s_1 \ldots s_n$, we know that only n-tuples of channel names with sorts s_i are sent or received via a channel with sort s. If a free name x and a bound name y have the same sort, we have to take into account that x may replace y during the reduction. We also use sorts as an intermediate level between names and monoid elements, since with α-conversion it is problematic to assign monoid elements directly to names.

Monoid elements appear in two places: in the range of m_Γ and in the tuples $x\colon \langle s/a\rangle$. The idea is to sum up the capabilities of x with $+$ in a while x is still free and add a to $m_\Gamma(s)$ with $\oplus$ as soon as x is hidden. We have to use $\oplus$ according to the explanation given in section 2.2. The other possibility would be to immediately add the capabilities to $m_\Gamma(s)$ with $\oplus$ (without storing them in a first), but since $a + b \leq a \oplus b$, this would lead to looser bounds. (It would, however, be possible in the case where we only consider monoid elements greater than or equal to 0, since in this case $+$ and $\oplus$ always coincide.)

In the rest of this paper we use the operations on type assignments given in table 2 (all operations on sequences are conducted pointwise): in (ADD-MON) we add a monoid element a to a type assignment Γ by adding a to $m_\Gamma(s)$ (with $\oplus$) and leaving everything else unchanged. And $\Gamma^{\langle \tilde{s}, \tilde{a}\rangle}$ denotes $(\ldots(\Gamma^{\langle s_1, a_1\rangle})^{\langle s_2, a_2\rangle}\ldots)^{\langle s_n, a_n\rangle}$. (ADD-CAP) describes how to add a monoid element corresponding to a name x. In this case b is added directly (with $+$) to the monoid elements assigned to names, rather than to those assigned to sorts.

$$\text{(ADD-MON)} \quad (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a}\rangle)^{\langle s,a\rangle} = ob; m'; \tilde{x} \colon \langle \tilde{s}/\tilde{a}\rangle$$
$$\text{where } m'(s') = \begin{cases} m(s) \oplus a & \text{if } s = s' \\ m(s) & \text{otherwise} \end{cases}$$

$$\text{(ADD-CAP)} \quad (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a}\rangle) + x_i : b = (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{c}\rangle)$$
$$\text{where } c_j = \begin{cases} a_i + b & \text{if } i = j \\ a_j & \text{otherwise} \end{cases}$$

$$\text{(SUM)} \quad (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a}\rangle) \circledast (ob; m'; \tilde{x} \colon \langle \tilde{s}/\tilde{b}\rangle) = ob; m \oplus m'; \tilde{x} \colon \langle \tilde{s}/\tilde{a} + \tilde{b}\rangle$$

$$\text{(JOIN)} \quad (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a}\rangle) \vee b = ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a} \vee b\rangle$$

$$\text{(ORD)} \quad (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a}\rangle) \le (ob; m'; \tilde{x} \colon \langle \tilde{s}/\tilde{b}\rangle) \iff m \le m' \text{ and } \tilde{a} \le \tilde{b}$$

$$\text{(REMOVE)} \quad \text{If } \Gamma = \Delta, x : \langle s/a\rangle, \text{ then } \Gamma \backslash \{x\} = \Delta$$

Table 2. Operations on type assignments

Summation $\Gamma \circledast \Gamma'$ (SUM) is defined for type assignments which contain the same names (having identical sorts) and which satisfy $ob_\Gamma = ob_{\Gamma'}$ (i.e. they have the same sort structure). In this case $m \oplus m'$ and $\tilde{a} + \tilde{b}$ denote the pointwise summation. The summation on type assignments has a counterpart (denoted by the same symbol) in Honda's work [6].

In (JOIN) a pointwise join with every monoid element assigned to a channel name and the monoid element b is defined. And finally we need a partial order on type assignments (ORD) and an operation removing an assumption on a name x from a type assignment (REMOVE).

The operations on type assignments satisfy the following laws:

Lemma 1. *It holds that*

$$\Gamma^{\langle s,a\rangle} \circledast \Delta = (\Gamma \circledast \Delta)^{\langle s,a\rangle} \qquad (\Gamma + x : a) \circledast \Delta = (\Gamma \circledast \Delta) + x : a$$
$$(\Delta, x \colon \langle s/a\rangle)^{\langle y,b\rangle} = \Delta^{\langle y,b\rangle}, x \colon \langle s, a\rangle \qquad \left(\Gamma^{\langle s,a\rangle}\right)^{\langle t,b\rangle} = \left(\Gamma^{\langle t,b\rangle}\right)^{\langle s,a\rangle}$$

Furthermore $\circledast$ is associative and commutative.

Proof. Straightforward from the definition of the operations.

We are now ready to define the rules of the type system (see table 3). *out* and *in* are fixed monoid elements (where *in* must be comparable to 0) representing the capabilities of output respectively input prefixes.

$$\frac{\Gamma \vdash P,\ \Gamma \leq \Delta}{\Delta \vdash P}\,(\text{T-}\leq) \qquad \Gamma \vee 0 \vdash \mathbf{0}\ (\text{T-NIL}) \qquad \frac{\Gamma_1 \vdash P_1,\ \Gamma_2 \vdash P_2}{\Gamma_1 \circledast \Gamma_2 \vdash P_1 \mid P_2}\,(\text{T-PAR})$$

$$\frac{\Gamma, x\colon \langle s/a\rangle, \tilde{z}\colon \langle \tilde{t}, \tilde{b}\rangle \vdash P}{(\Gamma, \tilde{z}\colon \langle \tilde{t}, \tilde{b}\rangle) \vee 0, x\colon \langle s/(a - \mathit{in}) \vee 0 + \mathit{out}\rangle \vdash \bar{x}\langle \tilde{z}\rangle.P}\,(\text{T-OUT}) \quad \text{if } ob_\Gamma(s) = \tilde{t}$$

$$\frac{\Gamma, x\colon \langle s/a\rangle, \tilde{y}\colon \langle \tilde{t}/\tilde{b}\rangle \vdash P}{(\Gamma \vee 0, x\colon \langle s/(a - \mathit{out}) \vee 0 + \mathit{in}\rangle)^{\langle \tilde{t}, \tilde{b}\rangle} \vdash x(\tilde{y}).P}\,(\text{T-IN}) \quad \text{if } ob_\Gamma(s) = \tilde{t}$$

$$\frac{\Gamma \backslash \{x\}, x\colon \langle s/a\rangle \vdash P}{\Gamma^{\langle x, a\rangle} \vdash (\nu x\colon s)P}\,(\text{T-RESTR}) \qquad \frac{\Gamma \vdash x(\tilde{y}).P}{\Delta, x\colon \langle s/a + \mathit{sig}(\mathit{in})\rangle \vdash !x(\tilde{y}).P}\,(\text{T-REP})$$
$$\text{if } \Gamma \circledast \Gamma \leq \Gamma \text{ and } \Gamma = \Delta, x\colon \langle t/a\rangle$$

Table 3. Rules of the type system

The intuitive meaning of the rules is as follows:

(T-$\leq$) We can always over-approximate the capabilities of a process.

(T-NIL) The nil process can have an arbitrary type assignment, provided the monoid elements of the free names are greater than 0.

(T-PAR) The parallel composition of two processes can be typed by adding their respective type assignments.

(T-OUT) First we subtract *in* from the monoid element a associated with the subject x of the output prefix, since the emergence of P means the removal of an input prefix with subject x somewhere else in the environment. We then take the join of all monoid elements and 0, since we only consider capabilities on the outer level of processes and thus we only consider future influence by positive capabilities, but not by negative ones (since we are doing over-approximation). In the end *out* is added to the monoid element associated with x.

Furthermore we have to check that the sort structure is correct, i.e. since $z_1, \ldots, z_n$ are communicated via x, the string of their sorts must be the object sort of the sort of x.

(T-IN) As described for (T-OUT), we subtract *out*, take the join with 0 and then add *in*. Furthermore we check the correctness of the sort structure as above.

Since, in this case, $y_1, \ldots, y_n$ are bound by the input prefix, all assumptions on these names are removed from the type assignment, and their monoid elements are added to the rest of the type assignment with $\oplus$.

(T-RESTR) If a name is hidden, we remove the assumption on it, but retain information on its capabilities by adding its monoid element to the type assignment and by keeping the sort.

(T-REP) In this rule we have to make sure that a replicated process has a type assignment which is either idempotent or gets smaller when added to itself. This can be achieved if Γ contains only negative or idempotent monoid elements.
And furthermore, since we know that infinitely many copies of the input prefix with subject x are available, we add $\bot$, $\top$ or 0, according to the value of *in*.

The type system satisfies the following substitution lemma, which is central for proving the subject reduction property:

Lemma 2. (Substitution) *Let $x \neq y$ be two names.*
If $\Sigma = (\Gamma, x\colon \langle s/a\rangle, y\colon \langle s/b\rangle) \vdash P$, then $\Sigma' = (\Gamma, x\colon \langle s/a+b\rangle) \vdash P\{x/y\}$.

Proof. By induction on the typing rules:

(T-**0**) $P = \mathbf{0}$ and it follows that $a \geq 0$ and $b \geq 0$ and thus also $a + b \geq 0$. (T-**0**) implies that $\Gamma, x\colon \langle s/a+b\rangle \vdash P$.

(T-$\leq$) $\Upsilon \vdash P$ and $\Upsilon \leq \Sigma$. We can assume that $\Upsilon = \Gamma', x\colon \langle s/a'\rangle, y\colon \langle s/b'\rangle$ where $\Gamma' \leq \Gamma$, $a' \leq a$ and $b' \leq b$. It follows with the induction hypothesis that $\Gamma', x\colon \langle s/a'+b'\rangle \vdash P\{x/y\}$, and this type assignment is smaller (in the $\leq$-relation) than $\Gamma, x\colon \langle t/a+b\rangle = \Sigma'$ and thus $\Sigma' \vdash P\{x/y\}$.

(T-PAR) $P = P_1 \mid P_2$, that is $P\{x/y\} = P_1\{x/y\} \mid P_2\{x/y\}$ and

$$\Gamma_i, x\colon \langle s/a_i\rangle, y\colon \langle s/b_i\rangle \vdash P_i,\ i \in \{1,2\}$$
$$\Sigma = \Gamma_1 \circledast \Gamma_2, x\colon \langle s/a_1+a_2\rangle, y\colon \langle s/b_1+b_2\rangle$$

with $\Gamma = \Gamma_1 \circledast \Gamma_2$, $a = a_1 + a_2$ and $b = b_1 + b_2$. From the induction hypothesis it follows that $\Gamma_i, x\colon \langle s/a_i + b_i\rangle \vdash P_i\{x/y\}$. This implies that $\Sigma' = \Gamma_1 \circledast \Gamma_2, x\colon \langle s/a_1+a_2+b_1+b_2\rangle \vdash P\{x/y\}$.

(T-OUT) $P = \bar{z}\langle\tilde{z}\rangle.Q$, that is

$$\frac{\Delta, z\colon \langle t/a'\rangle \vdash Q}{\Sigma = \Delta \vee 0, z\colon \langle t/(a' - \mathit{in}) \vee 0 + \mathit{out}\rangle \vdash P}$$

and $\mathit{ob}_\Sigma(t) = \mathit{ob}_\Sigma(\mathit{sort}_\Sigma(\tilde{z}))$.
Now let σ be a substitution which replaces y by x and is constant on all other names. Since the sorts of x and y are equal it holds that $\mathit{sort}_{\Sigma'}(\sigma(w)) = \mathit{sort}_\Sigma(w)$ for all names in Σ.
It thus follows that $\mathit{ob}_{\Sigma'}(\mathit{sort}_{\Sigma'}(\sigma(z))) = \mathit{ob}_\Sigma(\mathit{sort}_\Sigma(z)) = \mathit{ob}_\Sigma(t) = \mathit{ob}_\Sigma(\mathit{sort}_\Sigma(\tilde{z})) = \mathit{ob}_{\Sigma'}(\mathit{sort}_{\Sigma'}(\sigma(\tilde{z})))$ and thus the condition for the typing of $P\{x/y\} = \overline{\sigma(z)}\langle\sigma(\tilde{z})\rangle.(Q\{x/y\})$ is satisfied.
We now distinguish the following cases:

$z \neq x \wedge z \neq y$**:** In this case $\Delta = \Delta', x\colon \langle s/a\rangle, y\colon \langle s/b\rangle$. It follows with the induction hypothesis that $\Delta', x\colon \langle s/a+b\rangle, z\colon \langle t/a'\rangle \vdash Q\{x/y\}$. And we can conclude that

$$\Delta' \vee 0, x\colon \langle s/(a+b) \vee 0\rangle, z\colon \langle t/(a' - \mathit{in}) \vee 0 + \mathit{out}\rangle \vdash P\{x/y\}$$

Since $(a+b) \vee 0 \leq a \vee 0 + b \vee 0$ (see section 2.2) this type assignment smaller (in the $\leq$-relation) than

$$\Sigma' = \Delta' \vee 0, x\colon \langle s/a \vee 0 + b \vee 0\rangle, z\colon \langle t/(a' - \mathit{in}) \vee 0 + \mathit{out}\rangle$$

And with (T-$\leq$) it follows that $\Sigma' \vdash P\{x/y\}$

$z = x \wedge z \neq y$**:** In this case $\Delta = \Delta', y\colon \langle s/b\rangle$, $t = s$ and $a = a'$. It follows from the induction hypothesis that $\Delta', x\colon \langle s/a + b\rangle \vdash Q\{x/y\}$ and $\Upsilon = \Delta' \vee 0, x\colon \langle s/((a+b) - \mathit{in}) \vee 0 + \mathit{out}\rangle \vdash P\{x, y\}$.

On the other hand $\Sigma = \Delta' \vee 0, y\colon \langle s/b \vee 0\rangle, x\colon \langle s/(a - \mathit{in}) \vee 0 + \mathit{out}\rangle$ and $\Sigma' = \Delta' \vee 0, x\colon \langle s/(a - \mathit{in}) \vee 0 + \mathit{out} + b \vee 0\rangle$. And since (see section 2.2)

$$((a+b) - \mathit{in}) \vee 0 + \mathit{out} \leq ((a - \mathit{in}) + b) \vee 0 + \mathit{out}$$
$$\leq (a - \mathit{in}) \vee 0 + b \vee 0 + \mathit{out}$$

the type assignment Υ is smaller than Σ' and (T-$\leq$) implies $\Sigma' \vdash P\{x/y\}$.

$z = x \wedge z \neq y$**:** this is analogous to the case above.

(T-IN) $P = z(\tilde{z}).Q$, that is

$$\frac{\Delta, z\colon \langle t/a'\rangle, \tilde{z}\colon \langle \tilde{t}, \tilde{c}\rangle \vdash Q}{\Sigma = (\Delta \vee 0, z\colon \langle t/(a' - \mathit{out}) \vee 0 + \mathit{in}\rangle)^{\langle \tilde{t}, \tilde{c}\rangle} \vdash P}$$

and $ob_\Sigma(t) = \tilde{t}$.

Now let σ be again a substitution which replaces y by x and is constant on all other names. Since the sorts of x and y are equal it holds that $\mathit{sort}_{\Sigma'}(\sigma(w)) = \mathit{sort}_\Sigma(w)$ for all names in Σ.

It thus follows that $ob_{\Sigma'}(\mathit{sort}_{\Sigma'}(\sigma(z))) = ob_\Sigma(\mathit{sort}_\Sigma(z)) = ob_\Sigma(t) = \tilde{t}$ and thus the condition for the typing of $P\{x/y\} = \sigma(z)(\tilde{z}).(Q\{x/y\})$ is satisfied.

We can assume that x, y and z are different from all names in $\tilde{z}$. We now distinguish the following cases:

$z \neq x \wedge z \neq y$**:** In this case $\Delta = \Delta', x\colon \langle s/a\rangle, y\colon \langle s/b\rangle$. It follows with the induction hypothesis that

$$\Delta', x\colon \langle s/a + b\rangle, z\colon \langle t/a'\rangle, \tilde{z}\colon \langle \tilde{t}/\tilde{c}\rangle \vdash Q\{x/y\}$$

And we can conclude that

$$(\Delta' \vee 0, x\colon \langle s/(a+b) \vee 0\rangle, z\colon \langle t/(a' - \mathit{out}) \vee 0 + \mathit{in}\rangle)^{\langle \tilde{t}, \tilde{c}\rangle} \vdash P\{x/y\}$$

Since $(a+b) \vee 0 \leq a \vee 0 + b \vee 0$ (see section 2.2) this type assignment smaller (in the $\leq$-relation) than

$$\Sigma' = (\Delta' \vee 0, x\colon \langle s/a \vee 0 + b \vee 0\rangle, z\colon \langle t/(a' - \mathit{out}) \vee 0 + \mathit{in}\rangle)^{\langle \tilde{t}, \tilde{c}\rangle}$$

And with (T-$\leq$) it follows that $\Sigma' \vdash P\{x/y\}$

$z = x \wedge z \neq y$**:** In this case $\Delta = \Delta', y\colon \langle s/b\rangle$, $t = s$ and $a = a'$. It follows with the induction hypothesis that

$$\Delta', x\colon \langle s/a + b\rangle, \tilde{z}\colon \langle \tilde{t}/\tilde{c}\rangle \vdash Q\{x/y\}$$
$$\Upsilon = (\Delta' \vee 0, x\colon \langle t/((a+b) - \mathit{out}) \vee 0 + \mathit{in}\rangle)^{\langle \tilde{t}, \tilde{c}\rangle} \vdash P\{x, y\}$$

On the other hand

$$\Sigma = \Delta' \vee 0, y\colon \langle s/b \vee 0\rangle, x\colon \langle s/(a - \mathit{out}) \vee 0 + \mathit{in}\rangle, \tilde{z}\colon \langle \tilde{t}/\tilde{c}\rangle$$
$$\Sigma' = (\Delta' \vee 0, x\colon \langle s/(a - \mathit{out}) \vee 0 + \mathit{in} + b \vee 0\rangle)^{\langle \tilde{t}, \tilde{c}\rangle}$$

And since (see section 2.2)

$$((a+b)-\mathit{out})\vee 0+\mathit{in}\leq((a-\mathit{out})+b)\vee 0+\mathit{in}$$
$$\leq(a-\mathit{out})\vee 0+b\vee 0+\mathit{in}$$

the type assignment Υ is smaller than Σ' and (T-$\leq$) implies that $\Sigma' \vdash P\{x/y\}$.

$z=x\wedge z\neq y$**:** this is analogous to the case above.

(T-RESTR) $P=(\nu z\colon t)Q$, that is $\Delta\backslash\{z\}, z\colon\langle t/a'\rangle \vdash Q$ and $\Sigma=\Delta^{\langle t,a'\rangle} \vdash P$. Since z is bound, we can assume that $z\neq x$ and $z\neq y$, thus Δ has the form $\Delta', x\colon\langle s/a\rangle, y\colon\langle s/b\rangle$. The induction hypothesis implies that

$$\Delta'\backslash\{z\}, x\colon\langle s/a+b\rangle, z\colon\langle t/a'\rangle \vdash Q\{x/y\}$$

Therefore $(\Delta', x\colon\langle s/a+b\rangle)^{\langle t,a'\rangle} \vdash P\{x/y\}$. This type assignment is the same as $(\Delta')^{\langle t,a\rangle}, x\colon\langle s/a+b\rangle=\Sigma'$

(T-REP) $P=!z(\tilde{z}).Q$, that is

$$\frac{\Delta, z\colon\langle t/a'\rangle \vdash z(\tilde{z}).Q}{\Sigma=\Delta, z\colon\langle t/a'+\mathit{sig}(\mathit{in})\rangle \vdash P}$$

$$(\Delta, z\colon\langle t/a'\rangle)\circledast(\Delta, z\colon\langle t/a'\rangle)\leq\Delta, z\colon\langle t/a'\rangle$$

We distinguish the following cases:

$z\neq x, z\neq y$**:** In this case $\Delta=\Delta', x\colon\langle s/a\rangle, y\colon\langle s/b\rangle$ and it follows from the induction hypothesis that

$$\begin{aligned}
&\Delta', x\colon\langle s/a+b\rangle, z\colon\langle t/a'\rangle \vdash (z(\tilde{z}).Q)\{x/y\}\\
&(\Delta', x\colon\langle s/a+b\rangle, z\colon\langle t/a'\rangle)\circledast(\Delta', x\colon\langle s/a+b\rangle, z\colon\langle t/a'\rangle)\\
=\;&\Delta'\circledast\Delta', x\colon\langle s/a+b+a+b\rangle, z\colon\langle t/a'+a'\rangle\\
\leq\;&\Delta', x\colon\langle s/a+b\rangle, z\colon\langle t/a'\rangle
\end{aligned}$$

And therefore

$$\Sigma'=\Delta', x\colon\langle s/a+b\rangle, z\colon\langle t/a'+\mathit{sig}(\mathit{in})\rangle \vdash P\{x/y\}$$

$z=x, z\neq y$**:** In this case $\Delta=\Delta', y\colon\langle s/b\rangle$, $t=s$ and $a=a'$. The induction hypothesis implies that

$$\Delta', x\colon\langle s/a+b\rangle \vdash (z(\tilde{z}).Q)\{x/y\}$$

As above we can conclude that

$$\Delta', x\colon\langle s/(a+b+\mathit{sig}(\mathit{in}))\rangle \vdash P\{x/y\}$$

$z\neq x, z=y$**:** Analogous to the second case

□

The types defined in table 3 are not yet invariant under reduction: rather than Γ, a modified type assignment $\overline{\Gamma}$ satisfies the subject reduction property.

Let $\Gamma = ob; m; \tilde{x}\colon \langle \tilde{s}/\tilde{a}\rangle$ and define $\overline{\Gamma} = (ob; m; \tilde{x}\colon \langle \tilde{s}/\tilde{0}\rangle)^{\langle \tilde{s},\tilde{a}\rangle}$. That is we add all monoid elements of the remaining free names to m with $\oplus$. Constructing $\overline{\Gamma}$ directly during the typing process does not seem to be possible, since we first have to sum up monoid elements with $+$ and then add them to the type tree with $\oplus$ the moment they are hidden.

In order to show the subject reduction property we need the following lemma:

Lemma 3. *If $\Gamma, x : \langle s/a\rangle \vdash P$ and $x \notin fn(P)$ it follows that $a \geq 0$ and $\Gamma \vdash P$. And furthermore if $\Gamma \vdash P$ and Γ does not contain a name x, then $\Gamma, x : \langle s/a\rangle \vdash P$ for any sort s and any monoid element $a \geq 0$ (weakening).*

Now we can state the subject reduction property in the following way:

Proposition 1. (Subject Reduction Property) *If $P \equiv Q$ and $\Sigma \vdash P$, then it holds also that $\Sigma \vdash Q$. And if $P \to P'$ and $\Sigma \vdash P$ then there exists a type assignment Σ' such that $\Sigma' \vdash P'$ and $\overline{\Sigma'} \leq \overline{\Sigma}$.*

Proof. We show the first half of the proposition by induction on the rules of structural congruence.

α**-conversion** Let $(\nu y\colon s)P \equiv (\nu x\colon s)(P\{x/y\})$. (We assume that $x \notin fn(P)$).
If $\Sigma \vdash (\nu y\colon s)P$ it follows that $\Sigma \geq \Gamma^{\langle s,a\rangle}$ and $\Gamma\backslash\{y\}, y\colon \langle s/a\rangle \vdash P$. From lemma 3 it follows that $\Gamma\backslash\{x,y\}, y\colon \langle s/a\rangle, x\colon \langle s/0\rangle \vdash P$. Lemma 2 then implies that $\Gamma\backslash\{x,y\}, x\colon \langle s/a+0\rangle \vdash P\{x/y\}$. It follows that $(\Gamma\backslash\{y\})^{\langle s,a\rangle} \vdash (\nu x\colon s)(P\{x/y\})$ and therefore $\Sigma \vdash (\nu x\colon s)(P\{x/y\})$ (if y is contained in Γ, it might be necessary to add an assumption on y according to lemma 3).
Since every α-conversion can be reverted by one or more α-conversions, the other direction follows immediately.

(C-COM) obvious, since $\circledast$ is commutative (see lemma 1)

(C-**0**) If $\Sigma \vdash P \mid \mathbf{0}$ it follows that $\Sigma \geq \Gamma \circledast (\Delta \vee 0)$ where $\Gamma \vdash P$ and Δ is an arbitrary type assignment. It holds that $\Gamma \leq \Gamma \circledast (\Delta \vee 0) \leq \Sigma$ and with (T-$\leq$) it follows that $\Sigma \vdash P$.
If, on the other hand, $\Sigma \vdash P$ where $\Sigma = ob; m; \tilde{x}\colon \langle \tilde{s}/\tilde{a}\rangle$, we can construct a type assignment $\Delta = ob; m'; \tilde{x}\colon \langle \tilde{s}/\tilde{0}\rangle$ where $m'(s) = \bot$ for every $s \in S$. Δ satisfies $\Delta \vdash \mathbf{0}$ and $\Sigma = \Sigma \circledast \Delta \vdash P \mid \mathbf{0}$.

(C-ASS) obvious with the associativity of $\circledast$ (see lemma 1)

(C-RESTR1) Let $(\nu x\colon s)(\nu y\colon t)P \equiv (\nu y\colon t)(\nu x\colon s)P$ where $x \neq y$.
If $\Sigma \vdash (\nu x\colon s)(\nu y\colon t)P$ it follows that $\Sigma \geq (\Gamma^{\langle t,b\rangle})^{\langle s,a\rangle}$ and $\Gamma\backslash\{y,x\}, y\colon \langle t/b\rangle, x\colon \langle s/a\rangle \vdash P$. We can interchange x and y (see also lemma 1) and thus $\Sigma = (\Gamma^{\langle s,a\rangle})^{\langle t,b\rangle}$ and $\Gamma\backslash\{x,y\}, x\colon \langle s/a\rangle, y\colon \langle t/b\rangle \vdash P$.
Therefore $\Sigma \vdash (\nu y\colon t)(\nu x\colon s)P$.

(C-RESTR2) Let $((\nu x\colon s)P_1) \mid P_2 \equiv (\nu x\colon s)(P_1 \mid P_2)$ where $x \notin fn(P_2)$.
If $\Sigma \vdash ((\nu x\colon s)P_1) \mid P_2$ it follows that $\Sigma \geq \Gamma_1 \circledast \Gamma_2$ where $\Gamma_2 \vdash P_2$, $\Gamma_1 \geq \Gamma^{\langle s,a\rangle}$ and $\Gamma\backslash\{x\}, x\colon \langle s/a\rangle \vdash P_1$. Since $x \notin fn(P_2)$ it follows with lemma 3 that $\Gamma_2\backslash\{x\}, x : \langle s/0\rangle \vdash P_2$. Therefore $(\Gamma \circledast \Gamma_2)\backslash\{x\}, x\colon \langle s/a+0\rangle \vdash P_1 \mid P_2$ (Γ

contains an x if and only if Γ_2 contains an x). Thus $(\Gamma \circledast \Gamma_2)^{\langle s,a\rangle} \vdash (\nu x\colon s)(P_1 \mid P_2)$ and $(\Gamma \circledast \Gamma_2)^{\langle s,a\rangle} = \Gamma^{\langle s,a\rangle} \circledast \Gamma_2 \le \Gamma_1 \circledast \Gamma_2 \le \Sigma$. Then (T-$\le$) implies that $\Sigma \vdash (\nu x\colon s)(P_1 \mid P_2)$.

If, on the other hand, $\Sigma \vdash (\nu x\colon s)(P_1 \mid P_2)$ we can conclude that $\Sigma \ge \Gamma^{\langle s,a\rangle}$, $\Gamma\backslash\{x\}, x\colon \langle s/a\rangle \ge \Gamma_1 \circledast \Gamma_2$ and $\Gamma_i = \Gamma_i', x\colon \langle s/a_i\rangle \vdash P_i$, $i \in \{1,2\}$. Since $x \notin \mathit{fn}(P_2)$ it follows with lemma 3 that $a_2 \ge 0$ and therefore $a \ge a_1 + a_2 \ge a_1 + 0 = a_1$. It follows that $(\Gamma_1')^{\langle s,a_1\rangle} \vdash (\nu x\colon s)P_1$. Since it holds that $\Gamma_2' \vdash P_2$ (see lemma 3), we conclude that $(\Gamma_1')^{\langle s,a_1\rangle} \circledast \Gamma_2' = (\Gamma_1' \circledast \Gamma_2')^{\langle s,a_1\rangle} \vdash ((\nu x\colon s)P_1) \mid P_2$. If Σ does not contain the name x it follows that $\Sigma \ge (\Gamma_1' \circledast \Gamma_2')^{\langle s,a_1\rangle}$ and therefore $\Sigma \vdash ((\nu x\colon s)P_1) \mid P_2$. In the other case we have to add an assumption on x according to lemma 3.

closure under context: this follows immediately since typing is compositional

---

The second statement is shown by induction on the reduction rules. For the induction we need a stronger hypothesis than the one stated above. We show that if $\Sigma \vdash P$ and $P \to P'$ then there are type assignments Σ', Δ, a sequence $\tilde{x}$ of names occurring in Δ and a sequence of monoid elements $\tilde{a}$ of equal length such that

$$\Sigma' \vdash P' \qquad \Sigma' \le \Delta + \tilde{x}\colon \tilde{a} \qquad \Delta^{\langle \mathit{sort}_\Delta(\tilde{x}),\tilde{a}\rangle} \le \Sigma \tag{1}$$

Having shown (1) we can conclude that

$$\overline{\Sigma'} \le \overline{\Delta + \tilde{x}\colon \tilde{a}} \le \overline{\Delta^{\langle \mathit{sort}_\Delta(\tilde{x}),\tilde{a}\rangle}} \le \overline{\Sigma}$$

(The middle inequality holds since $b \oplus (c + a) \le (b \oplus a) \oplus c$.) It is left to show that (1) holds:

(R-COMM) Let $P = \bar{x}\langle\tilde{z}\rangle.Q \mid x\tilde{y}.R$, $P' = Q \mid R\{\tilde{z}/\tilde{y}\}$ and $\Sigma \vdash P$. This implies the use of the following typing rules[1]:

$$\frac{\Gamma, \tilde{z}\colon \langle\tilde{t}/\tilde{b}\rangle, x\colon \langle s/a\rangle \vdash Q}{\Gamma_1 = \Gamma \vee 0, \tilde{z}\colon \langle\tilde{t}/\tilde{b} \vee 0\rangle, x\colon \langle s/(a - \mathit{out}) \vee 0 + \mathit{in}\rangle \vdash \bar{x}\langle\tilde{z}\rangle.Q}$$

$$\frac{\Upsilon, \tilde{z}\colon \langle\tilde{u}/\tilde{c}\rangle, \tilde{y}\colon \langle\tilde{v},\tilde{a}\rangle, x\colon \langle s'/a'\rangle \vdash R}{\Gamma_2 = (\Upsilon \vee 0, \tilde{z}\colon \langle\tilde{u}/\tilde{c} \vee 0\rangle, x\colon \langle s'/(a' - \mathit{in}) \vee 0 + \mathit{out}\rangle)^{\langle\tilde{v},\tilde{a}\rangle} \vdash x(\tilde{y}).R}$$

And we know that $\mathit{ob}_\Gamma(s) = \tilde{t}$ and $\mathit{ob}_\Upsilon(s') = \tilde{v}$. And since $\Sigma \ge \Gamma_1 \circledast \Gamma_2$ is defined, it follows that $s = s'$ and $\tilde{t} = \tilde{u}$.

$$\begin{aligned}\Sigma \ge (\Gamma \vee 0) \circledast (\Upsilon^{\langle\tilde{v},\tilde{a}\rangle} \vee 0), \tilde{z}\colon \langle\tilde{t}/(\tilde{b} \vee 0) + (\tilde{c} \vee 0)\rangle,\\ x\colon \langle s/(a - \mathit{out}) \vee 0 + \mathit{in} + (a' - \mathit{in}) \vee 0 + \mathit{out}\rangle\end{aligned}$$

We know that

$$(\Gamma \vee 0) \circledast (\Upsilon^{\langle\tilde{v},\tilde{a}\rangle} \vee 0) \ge \Gamma \circledast \Upsilon^{\langle\tilde{v},\tilde{a}\rangle} \qquad (\tilde{b} \vee 0) + (\tilde{c} \vee 0) \ge \tilde{b} + \tilde{c}$$

$$\begin{aligned}&(a - \mathit{out}) \vee 0 + \mathit{in} + (a' - \mathit{in}) \vee 0 + \mathit{out} \ge\\ &\qquad (a - \mathit{out}) + \mathit{out} + (a' - \mathit{in}) + \mathit{in} \ge a + a'\end{aligned}$$

[1] We assume that $x \ne z_j$ for any j. In the other case, the proof has to be modified, but can be conducted analogously.

Thus

$$\Sigma \geq \Gamma \circledast \Upsilon^{\langle \tilde{v}, \tilde{a} \rangle}, \tilde{z} \colon \langle \tilde{t}/\tilde{b} + \tilde{c} \rangle, x \colon \langle s/a + a' \rangle$$

Since $ob_\Gamma = ob_\Upsilon$ it follows that $\tilde{u} = \tilde{t} = ob_\Gamma(s) = ob_\Upsilon(s') = \tilde{v}$ and the substitution lemma (lemma 2) is applicable which implies:

$$\Upsilon, \tilde{z} \colon \langle \tilde{t}/\tilde{a} + \tilde{c} \rangle, x \colon \langle s/a' \rangle \ \vdash \ R\{\tilde{z}/\tilde{y}\}$$

Thus $\Sigma' \ \vdash \ Q \mid R\{\tilde{z}/\tilde{y}\}$ with

$$\Sigma' = \Gamma \circledast \Upsilon, \tilde{z} \colon \langle \tilde{t}/\tilde{a} + \tilde{b} + \tilde{c} \rangle, x \colon \langle s/a + a' \rangle$$

We now set $\Delta = \Gamma \circledast \Upsilon, \tilde{z} \colon \langle \tilde{t}/\tilde{b} + \tilde{c} \rangle, x \colon \langle s/a + a' \rangle$ and show that condition (1) is satisfied: we use $\tilde{z}$ as the sequence of names and $\tilde{a}$ as the sequence of monoid elements.

$$\Delta^{\langle sort_\Delta(\tilde{z}), \tilde{a} \rangle} = \Delta^{\langle \tilde{v}, \tilde{a} \rangle} = \Gamma \circledast \Upsilon^{\langle \tilde{v}, \tilde{a} \rangle}, \tilde{z} \colon \langle \tilde{t}/\tilde{b} + \tilde{c} \rangle, x \colon \langle s/a + a' \rangle \leq \Sigma$$
$$\Sigma' = \Gamma \circledast \Upsilon, \tilde{z} \colon \langle \tilde{t}/\tilde{a} + \tilde{b} + \tilde{c} \rangle, x \colon \langle s/a + a' \rangle = \Delta + \tilde{z} : \tilde{a}$$

(R-PAR) Let $P = Q|R$, $Q \to Q'$, $P' = Q'|R$ and $\Sigma \ \vdash \ P$. In this case $\Gamma \ \vdash \ Q$, $\Upsilon \ \vdash \ R$ and $\Sigma \geq \Gamma \circledast \Upsilon$. The induction hypothesis implies that there are type assignments Δ_1, Γ' such that

$$\Gamma' \ \vdash \ Q' \qquad \Gamma' \leq \Delta_1 + \tilde{x} : \tilde{a} \qquad \Delta_1^{\langle sort_{\Delta_1}(\tilde{x}), \tilde{a} \rangle} \leq \Gamma$$

It follows that $\Sigma' = \Gamma' \circledast \Upsilon$ is defined and $\Sigma' \ \vdash \ Q' \mid R$. Now set $\Delta = \Delta_1 \circledast \Upsilon$. It holds that

$$\Delta^{\langle sort_\Delta(\tilde{x}), \tilde{a} \rangle} = \Delta_1^{\langle sort_{\Delta_1}(\tilde{x}), \tilde{a} \rangle} \circledast \Upsilon \leq \Gamma \circledast \Upsilon \leq \Sigma$$
$$\Sigma' = \Gamma' \circledast \Upsilon \leq (\Delta_1 + \tilde{x} : \tilde{a}) \circledast \Upsilon = (\Delta_1 \circledast \Upsilon) + \tilde{x} : \tilde{a} = \Delta + \tilde{x} : \tilde{a}$$

(R-REP) Let $P = \bar{x}\langle \tilde{z} \rangle.Q \mid !x(\tilde{y}).R$, $P' = Q \mid R\{\tilde{z}/\tilde{y}\} \mid !x(\tilde{y}).R$ and $\Sigma \ \vdash \ P$.
We will first show that if $\Lambda \ \vdash !x(\tilde{y}).R$, then it holds also that $\Lambda \ \vdash \ x(\tilde{y}).R \mid !x(\tilde{y}).R$. If $\Lambda \vdash !x(\tilde{y}).R$, it holds that

$$\frac{\Gamma \ \vdash \ x(\tilde{y}).R}{\Delta, x \colon \langle s/a + sig(in) \rangle \ \vdash !x(\tilde{y}).R}$$

where $\Gamma \circledast \Gamma \leq \Gamma$, $\Gamma = \Delta, x \colon \langle s/a \rangle$ and $\Lambda \geq \Delta, x \colon \langle s/a + sig(in) \rangle = \Delta'$. It follows that $\Gamma \circledast \Delta' \ \vdash \ x(\tilde{y}).R \mid !x(\tilde{y}).R$ and

$$\Gamma \circledast \Delta' = (\Delta, x \colon s/a \rangle) \circledast (\Delta, x \colon \langle s/a + sig(in) \rangle)$$
$$= \Delta \circledast \Delta, x \colon \langle s/a + a + sig(in) \rangle \leq \Delta, x \colon \langle s/a + sig(in) \rangle \leq \Lambda$$

Therefore $\Lambda \ \vdash \ x(\tilde{y}).R \mid !x(\tilde{y}).R$.
We know that $\Sigma \geq \Upsilon \circledast \Lambda$ where $\Upsilon \ \vdash \ \tilde{x}\langle \tilde{z} \rangle.Q$ and $\Lambda \ \vdash !x(\tilde{y}).R$. As shown above it holds that $\Lambda \ \vdash \ x(\tilde{y}).R \mid !x(\tilde{y}).R$ and thus $\Sigma \geq \Upsilon \circledast \Lambda \ \vdash \ \tilde{x}\langle \tilde{z} \rangle.Q \mid x(\tilde{y}).R \mid !x(\tilde{y}).R$. From (R-COMM) and (R-PAR) we know that the proposition holds for reduction by communication and reduction underneath a parallel composition and it follows that there exists a type assignment Σ' such that $\Sigma' \ \vdash \ Q \mid R\{\tilde{z}/\tilde{y}\} \mid !x(\tilde{y}).R$ and condition (1) is satisfied.

(R-RESTR) Let $P = (\nu x\colon s)Q$, $P' = (\nu x\colon s)Q'$ and $Q \to Q'$. Furthermore

$$\frac{\Gamma\backslash\{x\}, x\colon \langle s/a\rangle \;\vdash\; Q}{\Gamma^{\langle s,a\rangle} \;\vdash\; (\nu x\colon s)Q}$$

We assume that Γ does not contain an assumption on x, i.e. $\Gamma\backslash\{x\} = \Gamma$. Otherwise we proceed with $\Gamma\backslash\{x\}$ instead of Γ and add the missing assumption later with lemma 3. We know that $\Sigma \geq \Gamma^{\langle s,a\rangle}$.
The induction hypothesis implies that there exist type assignments $\Delta_1, x\colon \langle s/b\rangle$ and $\Gamma', x\colon \langle s/c\rangle$ and sequences $\tilde{x}, \tilde{a}$ such that

$$\begin{array}{ll} \Gamma', x\colon \langle s/c\rangle \;\vdash\; Q' & \quad \Gamma', x\colon \langle s/c\rangle \leq (\Delta_1, x\colon \langle s/b\rangle) + \tilde{x} : \tilde{a} \\ (\Delta_1, x\colon \langle s/b\rangle)^{\langle \mathit{sort}_{\Delta_1}(\tilde{x}),\tilde{a}\rangle} \leq \Gamma, x\colon \langle s/a\rangle & \end{array}$$

Now (T-RESTR) implies that $\Sigma' = (\Gamma')^{\langle s,c\rangle} \;\vdash\; (\nu x\colon s)Q'$. We distinguish two cases:

- $(\forall i \in \{1,\ldots,n\} : x \neq x_i)$: in this case $c \leq b \leq a$. We define $\Delta = \Delta_1^{\langle s,b\rangle}$. Then

$$\begin{aligned} &\Delta^{\langle \mathit{sort}_{\Delta}(\tilde{x}),\tilde{a}\rangle} = (\Delta_1^{\langle \mathit{sort}_{\Delta_1}(\tilde{x}),\tilde{a}\rangle})^{\langle s,b\rangle} \leq \Gamma^{\langle s,b\rangle} \leq \Gamma^{\langle s,a\rangle} \\ &\Sigma' = (\Gamma')^{\langle s,c\rangle} \leq (\Delta_1 + \tilde{x}\colon \tilde{a})^{\langle s,c\rangle} \leq \Delta_1^{\langle s,b\rangle} + \tilde{x}\colon \tilde{a} = \Delta + \tilde{x}\colon \tilde{a} \end{aligned}$$

- $(x = x_j)$: in this case $(\tilde{x} : \tilde{a})_j$ denotes the sequence of the x_i where x_j has been dropped. $c \leq b + a_j$ and $b \leq a$. We define $\Delta = \Delta_1^{\langle s,b\oplus a_j\rangle}$. Since the j-th element of $\mathit{sort}_{\Delta_1}(\tilde{x})$ is s it holds that

$$\begin{aligned} \Delta^{\langle \mathit{sort}_{\Delta}(\tilde{x}),\tilde{a}\rangle_j} &= (\Delta_1^{\langle s,b\oplus a_j\rangle})^{\langle \mathit{sort}_{\Delta_1}(\tilde{x}),\tilde{a}\rangle_j} = (\Delta_1^{\langle \mathit{sort}_{\Delta_1}(\tilde{x}),\tilde{a}\rangle})^{\langle s,b\rangle} \\ &\leq \Gamma^{\langle s,a\rangle} \leq \Sigma \\ \Sigma' &= (\Gamma')^{\langle s,c\rangle} \leq (\Delta_1 + (\tilde{x} : \tilde{a})_j)^{\langle s,b+a_j\rangle} \leq \Delta_1^{\langle s,b\oplus a_j\rangle} + (\tilde{x} : \tilde{a})_j \\ &= \Delta + (\tilde{x} : \tilde{a})_j \end{aligned}$$

□

4 Using the Type System for Process Analysis

As in other type systems for mobile processes, a type guarantees absence of runtime errors which may appear in the form of arity mismatches in the communication rules (R-COMM) and (R-REP), but it also enables us to perform more detailed process analysis.

4.1 Process Capabilities

The aim of this paper is to construct type systems yielding useful results for the analysis and verification of parallel processes. In our case the generic type system gives information concerning structural properties of a process, especially concerning its input and output capabilities. We will now formally define the connection between the type of a process and its capabilities.

Let P be a process and let x be a free name occurring in P. We define P's *capability* wrt. x by adding the following monoid elements: for every use of x as an output port we add *out* and for every use of x as an input port we add *in*. Notice that we do not continue summation after prefixes (see table 4).

$$C_x(\mathbf{0}) = 0 \quad C_x(P \mid Q) = C_x(P) + C_x(Q)$$

$$C_x(\bar{z}\langle\tilde{y}\rangle.P) = \begin{cases} \mathit{out} \text{ if } x = z \\ 0 \quad \text{otherwise} \end{cases} \qquad C_x(z(\tilde{y}).P) = \begin{cases} \mathit{in} \text{ if } x = z \\ 0 \;\; \text{otherwise} \end{cases}$$

$$C_x(!z(\tilde{z}).P) = \begin{cases} \mathit{sig}(\mathit{in}) \text{ if } x = z \\ 0 \qquad \text{otherwise} \end{cases} \qquad C_x((\nu y\colon s)P) = \begin{cases} C_x(P) \text{ if } x \neq y \\ 0 \qquad \text{otherwise} \end{cases}$$

Table 4. Determining the capabilities of a process

Proposition 2. *If $\Gamma \vdash P$, $P \to P'$ and x is a free name of P it follows that $C_x(P') \leq m_{\overline{\Gamma}}(\mathit{sort}_\Gamma(x))$, i.e. we determine the sort of x and look up the corresponding monoid element in $\overline{\Gamma}$. And if P contains a subexpression $(\nu y\colon s')Q$ it follows that the capabilities of y will never exceed $m_{\overline{\Gamma}}(s')$.*

Proof. We have to show that if $\Gamma \vdash P$ and x is a free name of P, then $C_x(P) \leq m_{\overline{\Gamma}}(\mathit{sort}_\Gamma(x))$. Then for any successor P' of P this follows immediately from the subject reduction property (proposition 1).

This can easily be done by induction on P and the only problematic case is replication: if $\Sigma = \Gamma, z\colon \langle s/a\rangle \vdash !z(\tilde{z}).Q$, it follows from (T-REP) that $\Gamma', z\colon \langle s/b\rangle \vdash z(\tilde{z}).Q$, where $\Gamma \geq \Gamma'$ and $a \geq b + \mathit{sig}(\mathit{in})$. Since $\Gamma', z\colon \langle s/b\rangle$ was produced by typing rule (T-IN) (and maybe rule (T-$\leq$) it follows that $b \geq (a' - \mathit{in}) \vee 0 + \mathit{out}$ for some monoid element a'. This implies that $a \geq (a' - \mathit{out}) \vee 0 + \mathit{in} + \mathit{sig}(\mathit{in}) \geq \mathit{in} + \mathit{sig}(\mathit{in}) = \mathit{sig}(\mathit{in})$.

And therefore $C_z(P) = \mathit{sig}(\mathit{in}) \leq a \leq m_\Gamma(s) \oplus a = m_{\overline{\Sigma}}(s) = m_{\overline{\Sigma}}(\mathit{sort}_\Sigma(z))$. For all other names $x \neq z$ it is straightforward to show that $C_x(P) = 0 \leq m_{\overline{\Sigma}}(\mathit{sort}_\Sigma(x))$.

The properties we can derive are of the form: "it is *always* the case that $C_x(P) \leq a$". There are many other interesting properties of a different nature we would like to derive. In section 5 we give some examples describing how these properties can be handled.

4.2 Composition of Type Systems

Given two type systems checking certain capabilities of processes, it is not difficult to construct a type system computing upper bounds for tuples of capabilities. Let

$$T_i = (I_i, \mathit{out}_i, \mathit{in}_i, X_i), \quad i \in \{1, 2\}$$

be two type systems, where the I_i are lattice-ordered monoids, $\mathit{out}_i, \mathit{in}_i$ are the monoid elements representing output respectively input capabilities and the X_i are predicates on type assignments, selecting only types satisfying a certain property. Because of (T-$\leq$) a predicate is only sensible if $X_i(\Gamma)$ and $\Gamma \geq \Delta$ imply $X_i(\Delta)$. (*Example:* A predicate X_i is true for all type assignments guaranteeing that any channel is used as an input channel and as an output channel at most once.) We say that T_i checks a predicate Y_i on processes if $\Gamma \vdash P$ and $X_i(\Gamma)$ imply $Y_i(P)$. (In our example $Y_i(P)$ could state that P is confluent, see also section 5.3.)

Then $T = (I_1 \times I_2, (\mathit{out}_1, \mathit{out}_2), (\mathit{in}_1, \mathit{in}_2), X)$ checks the conjunction of Y_1 and Y_2 if $X(i_1, i_2) = X_1(i_1) \wedge X_2(i_2)$ and it checks the disjunction of Y_1 and Y_2 if $X(i_1, i_2) = X_1(i_1) \vee X_2(i_2)$. (All monoid operations on $I_1 \times I_2$ are conducted pointwise.)

4.3 Type Inference

In order to support our claim that the type system is useful for the automated analysis of processes, we roughly sketch a type inference algorithm, determining the smallest type (in the $\leq$ relation defined in section 3) of a process P, provided P has a type. In order to make sure that a smallest type exists, we impose the following condition on the l-monoid: for every monoid element $a \in I$ there is a smallest element a' such that $a \leq a'$ and $a' + a' \leq a'$ (the same must be true for the operation $\oplus$)[2].

The algorithm proceeds in two steps:

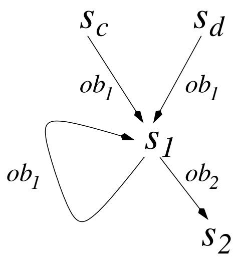
- In the first step we determine the assignment of sorts to names and the mapping ob_Γ. This may be done by representing *ob* as a graph and refining *ob* step by step by collapsing graph nodes every time we encounter a constraint of the form $\mathit{ob}(s) = \tilde{s}$. Or we can use the sort inference algorithm by Simon Gay [5].
- In the second step we compute the monoid elements by induction on the structure of P. In this case the typing rules are already very constructive, the main complication arises from typing rule (T-REP). Here we require that the monoid I satisfies the condition stated above. So (because of rule (T-$\leq$)) we may replace every monoid element a with its corresponding a' in the type assignment that we have derived so far.

A straightforward implementation of the algorithm has a runtime complexity quadratic in the size of P. Ameliorations are certainly possible by using efficient algorithms for unification and by finding an intelligent strategy for computing the monoid elements.

5 Examples

We now get back to the two example processes P and P' introduced in section 2.1 and type them with several instantiations of our type system, and thereby show how to mechanise process analysis in these cases.

[2] Every l-monoid useful for process analysis that we have come across so far satisfies this condition. In the case of $\mathbb{Z}^\infty$, a' is ∞ for positive a and a itself for all other elements.



We use the algorithm presented in section 4.3 to derive a type assignment Γ for P and P' and in the first step obtain a sort structure ob_Γ as shown in the figure to the left (ob_Γ is the same for P and P'). If there is an arrow labelled ob_i from sort s to sort t, then t is the i-th element of the sequence $ob_\Gamma(s)$. The assignment of names (in brackets we give the bound names) to sorts is:

$$c \colon s_c \quad (d \colon s_d) \quad h, i_1(, s, h_1, r) \colon s_1 \quad i_2(, h_2) \colon s_2$$

In the second step the monoid elements $m_{\overline{\Gamma}}(s)$ are computed (see below) in order to give an upper bound for all names having sort s.

5.1 Input/Output Behaviour of Channels

One simple application of our type system is to check whether channels are used for input, output or for both. We use the monoid IO (with elements *none*, *O*–"output only", *I*–"input only" and *both*) introduced in section 2.2. We set *in* $=$ *I*, *out* $=$ *O*.

For both processes P and P' we obtain the same type assignments with monoid elements shown in table 5 (row 1), i.e. i_2, h_2 are used neither for input nor output while all other names may be used for both. Note that, because of residuation, typing F alone yields capability *I* for name c, but no output capability. c acquires output capability only if communication with the environment is taking place.

This type system is similar to the one in [15] (apart from the fact that we consider types as a representation of process capabilities, rather than constraints on the environment), our type system however lacks a concept of co- and contravariance and thus our bounds are less tight.

5.2 Upper Bounds on the Number of Active Channels

We attempt to define a type system, similar to the one presented in [8] for our framework, i.e. we want to check how often a channel is used either for input or output.

We use the l-monoid $\mathbb{Z}^\infty \times \mathbb{Z}^\infty$ (cartesian product of the set of integers with ∞ and $-\infty$) introduced in section 2.2. The first component represents the number of active output prefixes (with a fixed subject) and the second component represents the number of active input prefixes.

We set *out* $= (1, 0)$, *in* $= (0, 1)$, and typing the processes P and P' yields the results given in table 5 (rows 2 & 3). Since for P the upper bound is always $(1, 1)$ or smaller we can conclude that there is at most one active input port and one active output port for any given subject at a time. For P' we can guarantee that, e.g. $\bar{c}$ always occurs at most once as an output prefix, although it occurs under a replication (see monoid element $m_{\overline{\Gamma}}(s_c)$).

5.3 Confluence

As in [8] we can use upper bounds on the number of active channels to guarantee confluence for π-calculus processes (see also [14]). Let Q be a process, and for every

	Property to be checked	$m_{\overline{T}}(s_d)$	$m_{\overline{T}}(s_c)$	$m_{\overline{T}}(s_1)$	$m_{\overline{T}}(s_2)$
1	Input/Output behaviour of P and P'	*both*	*both*	*both*	*none*
2	Upper bounds on active channels in P	$(1,1)$	$(1,1)$	$(1,1)$	$(0,0)$
3	Upper bounds on active channels in P'	(∞,∞)	$(1,\infty)$	$(1,\infty)$	$(0,0)$
4	Lower bounds on active channels in P	$(-1,0)$	$(-1,0)$	$(-1,-1)$	$(0,0)$
5	Lower bounds on active channels in P'	$(-\infty,\infty)$	$(-\infty,\infty)$	$(-\infty,-1)$	$(0,0)$
6	Avoiding blocked output prefixes in P'	(∞,∞)	$(1,\infty)$	$(1,-1)$	$(0,0)$

Table 5. Resulting monoid elements for different instantiations of the generic type system

name x in Q which is either free or bound by the scope operator ν it holds that its capabilities never exceed $(1,1)$. Then we can guarantee that every channel (also bound channels) occurs at most once at any given time as active input and output prefix, and we have non-overlapping redexes in (R-COMM). Thus we can conclude that if $Q \rightarrow^* Q'$, $Q' \rightarrow Q_1$ and $Q' \rightarrow Q_2$, then either $Q_1 \equiv Q_2$ or there is a process Q_3 such that $Q_1 \rightarrow Q_3$ and $Q_2 \rightarrow Q_3$.

Row 2 in table 5 provides upper bound $(1,1)$ for all capabilities in P. So we can state that P is confluent. Note that the same process would not be recognised as confluent by the type system in [8].

5.4 Lower Bounds on the Number of Active Channels

The type system is not limited to statements of the form: "there *at most* n active channels", we can also guarantee that there are *at least* m active channels. In order to achieve this, we use the type system above and just invert the partial order, i.e. we take $\geq$ instead of $\leq$, *out* and *in* remain unchanged. This means also that the join $\vee$ in the new partial order is now the meet $\wedge$ of the original partial order. Typing P does not give us much information, since we cannot guarantee that there are at least $m > 0$ prefixes active at any given time (see table 5, row 4) for any channel. In fact, some lower bounds are even (-1) stating that the respective channel removes input (or output) prefixes instead of making them available. In this case $P \rightarrow^* \mathbf{0}$ which means that no lower bounds can be guaranteed.

Typing P' yields the monoid elements given in table 5 (row 5) which states that input prefixes with subjects c, d are available infinitely often.

5.5 Avoiding Blocked Prefixes

Another interesting feature is to avoid blocked prefixes, i.e. prefixes which are waiting for a non-existing communication partner. We will first define—with the help of a lattice-ordered monoid—what it means for an output prefix to be blocked.

We take $\mathbb{Z}^\infty \times \mathbb{Z}^\infty$ as an l-monoid and define a new partial order: $(i,j) \sqsubseteq (i',j')$ iff $i \leq i'$ and $j \geq j'$. The first component represents the number of output prefixes and the second the number of input prefixes of the same subject. *out* $= (1,0)$ and *in* $= (0,1)$. We say a name x is *blocking* in P, if $P \rightarrow^* P'$, $C_x(P') \sqsupseteq (1,0)$ (i.e. there is at least one output prefix with subject x and no corresponding input prefixes) and for all P''

with $P' \to^* P''$ it follows that $C_x(P'') \sqsupseteq (1,0)$ (no communication with x will ever take place).

We can, e.g., avoid this situation, by demanding that it is always the case that $C_x(P') = (a,b)$ and either $a \leq 0$ or $b \geq 1$ (i.e. $(a,b) \not\sqsupseteq (1,0)$). We take the l-monoid and *out*, *in* introduced above. This type system can be obtained by composing a type system establishing upper bounds for input prefixes and one establishing lower bounds for output prefixes (see section 4.2). In this way we find out that all output prefixes with subjects c and d are non-blocking in P' (see table 5, row 6, where the tuples are composed out of the first component of the tuples in row 3 and the second component of the tuples in row 5).

This type system is not the only way to check for blocked prefixes. There are alternatives which can be employed in case this version fails.

6 Connection to Linear Types

A type system that has close connections to ours is the linear type system by Kobayashi, Pierce and Turner [8], since it also involves the typing of input/output capabilities of processes. "Linear" in this context means that the type system checks if a name is used once or several times for output respectively input.

The central aim of [8] is to introduce a new notion of barbed congruence by reducing the possible contexts of a process. This question has not been addressed in this paper, it is an interesting direction for future work. In the rest of this section we will highlight differences and interesting connections between the two type systems.

Our method of identifying linear channels is to establish upper bounds on the number of active channels and to check that these upper bounds are equal or smaller than one. Another approach would be to transform the type system in [8] directly into our framework, by using the capabilities presented there, i.e. form an l-monoid $\{|^1, !^1, ?^1, \updownarrow^1, |^\omega, !^\omega, ?^\omega, \updownarrow^\omega\}$ where $| = \emptyset$, $! = \{o\}$ (output), $? = \{i\}$ (input), $\updownarrow = \{i, o\}$. We define

$$p^x \vee q^y = (p \cup q)^{x \vee y} \text{ and } p^x + q^y = \begin{cases} (p \cup q)^{x \vee y} & \text{if } p \cap q = \emptyset \\ (p \cup q)^\omega & \text{otherwise} \end{cases}$$

The order on $\{1, \omega\}$ is $1 \leq \omega$. (The operations were defined partially in [8].) This is an l-monoid, which is, however, not residuated, since e.g. $|^\omega - ?^1$ is not defined. (The set $\{x \mid |^\omega \leq x + ?^1\}$ contains two minimal elements $?^1$ and $|^\omega$.)

As an alternative we use the l-monoid based on the set $\mathbb{Z}^\infty \times \mathbb{Z}^\infty$ (see section 5.2).

The two type systems differ in their definition of "linear channels": while in [8] a channel is considered to be linear if it is used at most once for input and output during the entire execution of a process, we consider a channel to be linear if at any given moment during the execution of a process it is active at most once for input and/or output. Thus our notion of "linear" encompasses the notion in [8].

Our notion of "linear" still implies confluence respectively partial confluence. The process P in our examples is identified as a confluent process (see section 5.3), while this would not be the case in the type system in [8].

The type system in [8] checks that output prefixes are always matched with a corresponding input prefix and vice versa. At first sight this resembles our method of avoiding blocked prefixes (see section 5.5). But since we only consider *active* prefixes the effects are different. While in [8] the process $Q = (\nu x \colon \updownarrow^1)(x.\bar{x}.\mathbf{0})$ can be typed, stating that x is linear and both input and output prefix are used, our type systems for avoiding blocked prefixes rejects this process, since it is actually blocked.

Our type system has one deficiency, in that it sometimes over-approximates too much. Consider, for example $Q = x \mid \bar{y} \mid \bar{z}\langle x\rangle \mid \bar{z}\langle y\rangle$. Following [8] this process is associated with the type: $z \ :!^{\omega}[\,|^1[\,]], x \ :?^1[\,], y \ :!^1[\,] \ \vdash \ Q$ whereas in our type system, we can only state that $\Gamma \vdash Q$ where $m_{\overline{\Gamma}}(sort_{\Gamma}(z)) = (2,0), m_{\overline{\Gamma}}(sort_{\Gamma}(x)) = m_{\overline{\Gamma}}(sort_{\Gamma}(y)) = (1,1)$, i.e. we have merged the upper bounds for x and y and have lost information, which means that we can no longer state that x is used only for input and w is used only for output.

This situation can probably be amended by introducing subtyping, i.e. covariant and contravariant types as in [15].

7 Conclusion and Future Work

This work has a similar aim as that of Honda [6], in that it attempts to describe a general framework for process analysis using type systems. We concentrate on a more specialised but still generic type system, which enables us to prove the subject reduction property for the general case. We have shown that, despite its generality, the type system can be instantiated in order to yield type systems related to existing ones. We have also shown how to parameterise type systems and what kind of parameters are feasible (in our case an l-monoid).

Our type system was derived from a type system for a graph-based process calculus with graphs as types, which make it easier to add additional behaviour information and which have a clear correspondence to associated monoid elements (via morphisms and categorical functors) [10]. A graph-based type system with lattices instead of monoids was presented in [11]. For lattices or positive cones of l-monoids, generic type systems are much easier to present. The main complication arises from non-positive elements and residuation.

Inspiration for this work came from papers deriving information on the behaviour of a process by inspecting its input/output capabilities, such as [15, 14, 8]. In order to conduct process analysis concerning more complex properties (as was done e.g. in [7, 4]) it is necessary to use type systems assigning behaviour information (i.e. monoid elements in our case) not only to single channels, but rather to tuples of channels or other more complex structures. This normally results in a semi-additive type system, in the terminology of Honda [6], while our present type system is strictly additive. In order to extend this type system, a first solution would be to allow monoid labels for n-ary tuples of names. Another idea is to integrate it into the categorical framework presented in [11], which would allow us to specify very general behaviour descriptions.

We believe that generic type systems can be developed into tools suitable for fast debugging and the analysis of concurrent programs. The next step is to apply the type system presented here to “real-life examples” and to more realistic programming languages.

Remark: this report is the extended version of [9].

Acknowledgements: I would like to thank the anonymous referees for their helpful comments, especially for the suggestion to use a sort system instead of type trees.

References

1. Martín Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, pages 611–638. Springer-Verlag, 1997.
2. Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
3. G. Birkhoff. *Lattice Theory*. American Mathematical Society, third edition, 1967.
4. Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the pi-calculus. In *Proc. of CONCUR ’98*.
5. Simon J. Gay. A sort inference algorithm for the polyadic π-calculus. In *Proc. of POPL ‘93*. ACM, 1993.
6. Kohei Honda. Composing processes. In *Proc. of POPL’96*, pages 344–357. ACM, 1996.
7. Naoki Kobayashi. A partially deadlock-free typed process calculus. In *Proc. of LICS ’97*, pages 128–139. IEEE, Computer Society Press, 1997.
8. Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. In *Proc. of POPL’96*, pages 358–371. ACM, 1996.
9. Barbara König. Analysing input/output-capabilities of mobile processes with a generic type system. In *Proc. of ICALP 2000*.
10. Barbara König. *Description and Verification of Mobile Processes with Graph Rewriting Techniques*. PhD thesis, Technische Universität München, 1999.
11. Barbara König. Generating type systems for process graphs. In *Proc. of CONCUR ’99*, pages 352–367. Springer-Verlag, 1999. LNCS 1664.
12. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992.
13. Robin Milner. The polyadic π-calculus: a tutorial. In F. L. Hamer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer-Verlag, Heidelberg, 1993.
14. Uwe Nestmann and Martin Steffen. Typing confluence. In *Second International ERCIM Workshop on Formal Methods in Industrial Critical Systems (Cesena, Italy, July 4–5, 1997)*, pages 77–101, 1997.
15. Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Proc. of LICS ‘93*, pages 376–385, 1993.
16. James Riely and Matthew Hennessy. Distributed processes and location failures. In *Proc. of ICALP’97*, pages 471–481. Springer-Verlag, 1997. LNCS 1256.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994
auf besondere Anforderung

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm
342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming
342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks
342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs
342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik
342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations
342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications
342/19/95 A Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication
342/21/95 A Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control
342/22/95 A Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes
342/23/95 A Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation
342/24/95 A M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems
342/01/96 A Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries
342/02/96 A Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs

Reihe A

342/03/96 A Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes
342/04/96 A Thomas Huckle: Efficient Computation of Sparse Approximate Inverses
342/05/96 A Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification
342/06/96 A Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components
342/07/96 A Richard Mayr: Some Results on Basic Parallel Processes
342/08/96 A Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht
342/09/96 A P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme
342/10/96 A Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments
342/11/96 A Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions
342/12/96 A Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete
342/13/96 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks
342/14/96 A Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project
342/15/96 A Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus
342/16/96 A Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen
342/17/96 A Richard Mayr: Model Checking PA-Processes
342/18/96 A Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States
342/01/97 A Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsf"ahigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken
342/02/97 A Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations
342/03/97 A Bernhard Mitschang: Technologie f"ur Parallele Datenbanken - Bericht zum Workshop
342/04/97 A nicht erschienen
342/05/97 A Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik
342/06/97 A Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump
342/07/97 A Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming
342/08/97 A Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement
342/09/97 A Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic
342/10/97 A Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus
342/11/97 A Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity
342/12/97 A Christian B. Czech: Architektur und Konzept des Dycos-Kerns
342/13/97 A Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow
342/14/97 A Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level
342/15/97 A Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen
342/16/97 A Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung

Reihe A

342/17/97 A Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing
342/18/97 A Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations
342/19/97 A Frank Wallner: Model Checking LTL Using Net Unfoldings
342/20/97 A Andreas Wolf, Andreas Kmoch: Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –
342/21/97 A Andreas Wolf, Marc Fuchs: Cooperative Parallel Automated Theorem Proving
342/22/97 A T. Ludwig, R. Wismüller, V. Sunderam, A. Bode: OMIS - On-line Monitoring Interface Specification (Version 2.0)
342/23/97 A Stephan Merkel: Verification of Fault Tolerant Algorithms Using PEP
342/24/97 A Manfred Broy, Max Breitling, Bernhard Schätz, Katharina Spies: Summary of Case Studies in Focus - Part II
342/25/97 A Michael Jaedicke, Bernhard Mitschang: A Framework for Parallel Processing of Aggregat and Scalar Functions in Object-Relational DBMS
342/26/97 A Marc Fuchs: Similarity-Based Lemma Generation with Lemma-Delaying Tableau Enumeration
342/27/97 A Max Breitling: Formalizing and Verifying TimeWarp with FOCUS
342/28/97 A Peter Jakobi, Andreas Wolf: DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)
342/29/97 A Radu Grosu, Ketil Stølen: Compositional Specification of Mobile Systems
342/01/98 A A. Bode, A. Ganz, C. Gold, S. Petri, N. Reimer, B. Schiemann, T. Schnekenburger (Herausgeber): "'Anwendungsbezogene Lastverteilung"', ALV'98
342/02/98 A Ursula Hinkel: Home Shopping - Die Spezifikation einer Kommunikationsanwendung in FOCUS
342/03/98 A Katharina Spies: Eine Methode zur formalen Modellierung von Betriebssystemkonzepten
342/04/98 A Stefan Bischof, Ernst-W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions
342/05/98 A St. Bischof, R. Ebner, Th. Erlebach: Load Balancing for Problems with Good Bisectors and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results
342/06/98 A Giannis Bozas, Susanne Kober: Logging and Crash Recovery in Shared-Disk Database Systems
342/07/98 A Markus Pizka: Distributed Virtual Address Space Management in the MoDiS-OS
342/08/98 A Niels Reimer: Strategien für ein verteiltes Last- und Ressourcenmanagement
342/09/98 A Javier Esparza, Editor: Proceedings of INFINITY'98
342/10/98 A Richard Mayr: Lossy Counter Machines
342/11/98 A Thomas Huckle: Matrix Multilevel Methods and Preconditioning
342/12/98 A Thomas Huckle: Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning
342/13/98 A Antonin Kucera, Richard Mayr: Weak Bisimilarity with Infinite-State Systems can be Decided in Polynomial Time
342/01/99 A Antonin Kucera, Richard Mayr: Simulation Preorder on Simple Process Algebras
342/02/99 A Johann Schumann, Max Breitling: Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern – Fallstudie –
342/03/99 A M. Bader, M. Schimper, Chr. Zenger: Hierarchical Bases for the Indefinite Helmholtz Equation
342/04/99 A Frank Strobl, Alexander Wisspeintner: Specification of an Elevator Control System
342/05/99 A Ralf Ebner, Thomas Erlebach, Andreas Ganz, Claudia Gold, Clemens Harlfinger, Roland Wism"uller: A Framework for Recording and Visualizing Event Traces in Parallel Systems with Load Balancing

Reihe A

342/06/99 A Michael Jaedicke, Bernhard Mitschang: The Multi-Operator Method: Integrating Algorithms for the Efficient and Parallel Evaluation of User-Defined Predicates into ORDBMS
342/07/99 A Max Breitling, Jan Philipps: Black Box Views of State Machines
342/08/99 A Clara Nippl, Stephan Zimmermann, Bernhard Mitschang: Design, Implementation and Evaluation of Data Rivers for Efficient Intra-Query Parallelism
342/09/99 A Robert Sandner, Michael Mauderer: Integrierte Beschreibung automatisierter Produktionsanlagen - eine Evaluierung praxisnaher Beschreibungstechniken
342/10/99 A Alexander Sabbah, Robert Sandner: Evaluation of Petri Net and Automata Based Description Techniques: An Industrial Case Study
342/01/00 A Javier Esparza, David Hansel, Peter Rossmanith, Stefan Schwoon: Efficient Algorithm for Model Checking Pushdown Systems
342/02/00 A Barbara König: Hypergraph Construction and Its Application to the Compositional Modelling of Concurrency
342/03/00 A Max Breitling and Jan Philipps: Verification Diagrams for Dataflow Properties
342/04/00 A Günther Rackl: Monitoring Globus Components with MIMO

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox - Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS