Trace Refinement of π-Calculus Processes*

Manuel Gieseking (manuel.gieseking@informatik.uni-oldenburg.de)

Correct System Design, Carl von Ossietzky University of Oldenburg, Germany

Classically, the general focus while investigating the π-calculus [4] is put on bisimulations between processes to determine their behavioral equivalence [5]. From a somewhat more application-oriented point of view, it is of interest to analyze process algebras in terms of the so-called *refinement* relation. That is to determine whether an implementation satisfies its specification. This approach has for instance been investigated in the context of Tony Hoare's CSP (Communication Sequential Processes [2]).

Both algebras are widely accepted for modeling and analyzing the communication between concurrent systems. Even though they have similar capabilities like parallel composition of processes, choice operators and actions/events which can be performed, they differ in some main features like the mobility aspect of the π-calculus to establish new communication by sending channels over channels.

Tony Hoare wrote in 2006 about CCS, the foundation of the π-calculus without the mobility aspect: "[...] the time has come to unify the two modelling styles [(CSP and CCS)], to enable practicing engineers to exploit a combination of their complementary advantages"[1]. Following this approach, we investigate in our work how the refinement notions of CSP can be applied to the π-calculus to exploit their advantages. Therefore, we present a new denotational semantics for the monadic π-calculus with guarded choice and without match and mismatch prefixes [1]. This approach is inspired by the trace semantics of CSP, as presented in [6] for example. Our further contributions are the proofs of compositionality results of the semantics, proofs establishing relations between trace refinement and (weak and strong) simulations, as well as algebraic laws and additional properties of the trace semantics for example concerning applications of substitutions and transpositions of names.

We choose the *early transition system* [7] as the foundation of our semantics, where every possible behavior a process can have is encoded in the label of the transitions. Thus, we can define a *big-step semantics*, similar to approaches for CSP, by collecting the labels of the transitions while following a path through the transition system. Those sequences of labels are also called *traces*. They hide the internal steps, that is, the internal communication of the process, since we would like to relate processes by their behavior visible to the environment. By collecting all possible traces of a process, we are obtaining the *trace semantics*. Instead of using the processes themselves we take their equivalence

*Acknowledgements: I would like to thank Sven Linker and Ernst-Rüdiger Olderog for being my supervisors and guides during the work of the underlying Master's thesis. Especially Sven, thank you for your patience and the hours of fruitful discussions.

[1] [3], page 209.

classes (denoted by $[P]$ for a process P) with respect to the renaming of bound names within the process. The *bound names* are those occurring as an *object* of an *input prefix* (for example the name x in the process $a(x).P$) or as a restricted name within a process (that is, for instance, the name x in $\underline{\texttt{new}}\, x\ P$). For a process P evolving with a big-step to Q by using the trace t, we write $[P] \stackrel{t}{\Rightarrow} [Q]$. The trace semantics for a process P is denoted by $\mathcal{T}([P])$ and a trace itself is written $t = \langle \alpha_1, \ldots, \alpha_n \rangle$, with only visible actions α_i. Thus, we can now define a condition (the specification) and check whether another process (the program) meets this specification by calculating the process' semantics and checking set-inclusion. We say a process Q *refines* P (written $P \sqsubseteq_{\mathcal{T}} Q$) if and only if $\mathcal{T}([Q]) \subseteq \mathcal{T}([P])$.

As an example we consider an abstract communication protocol, where Alice A sends a message m_1 to Bob B over a server S via a private channel c_1.

$$
\begin{aligned}
COM &=_{\text{def}} A \mid S \mid B \\
A &=_{\text{def}} \underline{\texttt{new}}\, c_1\, (\overline{s}\langle c_1 \rangle.\overline{c_1}\langle b \rangle.\overline{c_1}\langle m_1 \rangle) \\
S &=_{\text{def}} s(c).c(r).c(m).\left(\overline{log}\langle m \rangle.\overline{r}\langle m \rangle + \overline{r}\langle m \rangle\right) \\
B &=_{\text{def}} b(m_2).\overline{y}\langle m_2 \rangle
\end{aligned}
$$

First of all, Alice can establish a connection to the server via channel s by sending it the private channel c_1, which is further used for the communication between Alice and the server. Then Alice sends the name of the recipient b and the message m_1 over this private channel. Afterwards the server has the possibility to log the message and subsequently sending the received message to Bob or directly delivering it. Bob only listens at his address for some message and then does something not further specified with this message. This desired behavior is visualized through an extract of the process' operational semantics within Fig. 1. The internal communications are denoted by τ-transitions. The transitions which are used for calculating the visualized big-step with its trace $\langle \overline{log}\langle m_1 \rangle, \overline{b}\langle m_1 \rangle \rangle$, are the transitions of the left branch of the given extract of the transition system.

The trace semantics does not only consider the behavior resulting from the synchronized communication of the processes, but also collects all interleavings of the visible actions the constituent processes can perform. Figure 2 visualizes an extract of Alice's actions. Remark that the transitions labeled with $\overline{s}(x)$, where x may be any name, which does not occur free in the communication process, exist by reason of considering equivalence classes of the processes.

By collecting all of those traces – meaning all paths and all of their prefixes – in one set, we are gaining the trace semantics. That is for a process P the trace semantics is defined by

$$
\mathcal{T}([P]) = \left\{ t \in \texttt{Traces} \;\mid\; \exists Q \in \mathcal{P}^{\pi} : [P] \stackrel{t}{\Rightarrow} [Q] \right\},
$$

where $\mathcal{P}^{\pi}$ is the set of all processes and `Traces` the set of all traces. If we additionally consider a server without logging ($S' =_{\text{def}} s(c).c(r).c(m).\overline{r}\langle m \rangle$), we can prove by set-inclusion that the protocol without logging refines the other, since it has less behavior than the logging one.

The compositionality of the CSP trace semantics is one advantage which we tried to preserve by the definition of our denotational semantics. That is, for each syntactical

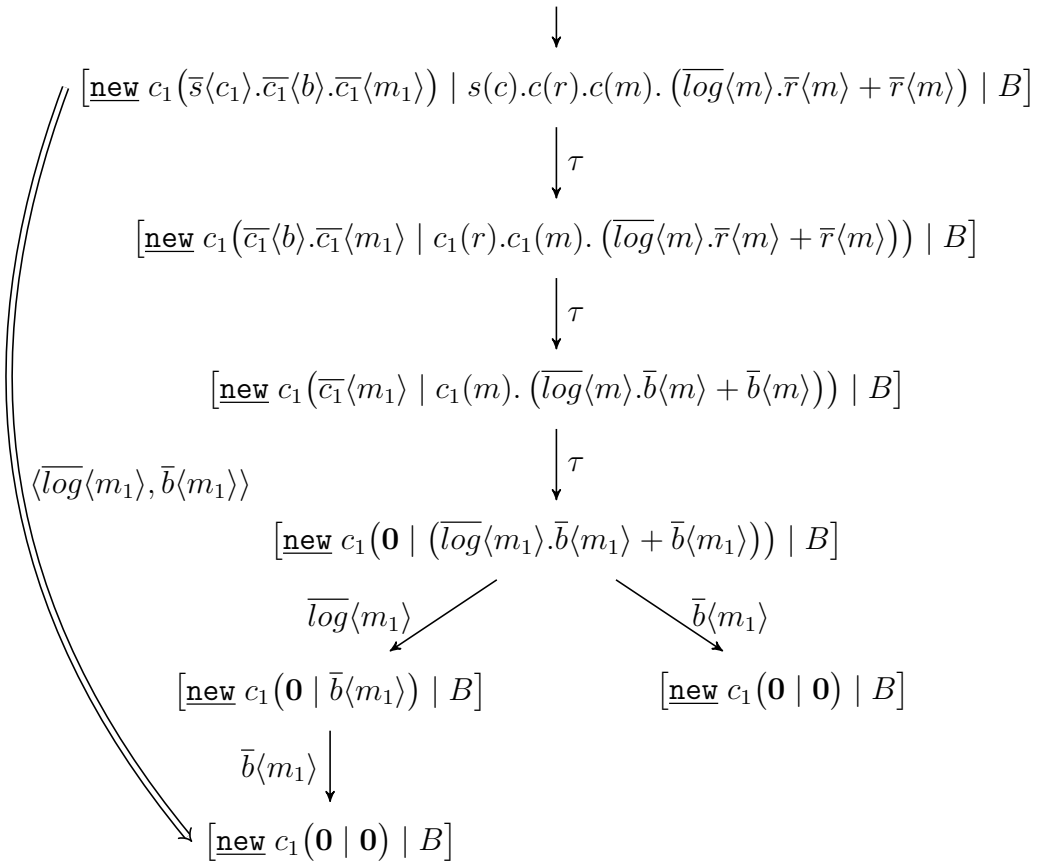

Figure 1: An extract of the early transition system of the communication between Alice and the server.

operator of the π-calculus we would like to have a semantical counterpart, such that the semantics of the composition of the syntactical processes is equal to the composition of the semantics of the syntactically composed parts. For instance, we try to find for a binary syntactical operator $\circ$ a semantical one $\circ_{\mathcal{T}}$ with $\mathcal{T}([P \circ Q]) = \mathcal{T}([P]) \circ_{\mathcal{T}} \mathcal{T}([Q])$. We show that the τ and output prefix, as well as the choice operator are compositional similar to their corresponding operators in CSP. Thus, we can easily show that for given processes $P, Q \in \mathcal{P}^{\pi}$ and names $a, x \in \mathcal{N}$

$$\mathcal{T}([\tau.P]) = \mathcal{T}([P]) \tag{TAU}$$
$$\mathcal{T}([\overline{a}\langle x\rangle.P]) = \{\langle\rangle\} \cup \{\langle\overline{a}\langle x\rangle\rangle\} \frown \mathcal{T}([P]) \tag{OUTPUT}$$
$$\mathcal{T}([P + Q]) = \mathcal{T}([P]) \cup \mathcal{T}([Q]) \tag{CHOICE}$$

hold. Where $\frown$ is the concatenation of the traces within the given sets, $\langle\rangle$ the empty trace without containing any visible behavior, and $\mathcal{N}$ the set of all possible names / channels. But due to the specialty of the π-calculus permitting new communication to be established by transmitting a channel via an input process, the compositionality of the input prefix is limited. Given a process $P \in \mathcal{P}^{\pi}$ and names $a, x \in \mathcal{N}$, then

$$\mathcal{T}([a(x).P]) = \{\langle\rangle\} \cup \{\langle a\, y\rangle \frown s \mid s \in \mathcal{T}([P\,\{{}^{y}/_{x}\}]), y \in \mathcal{N}\}$$

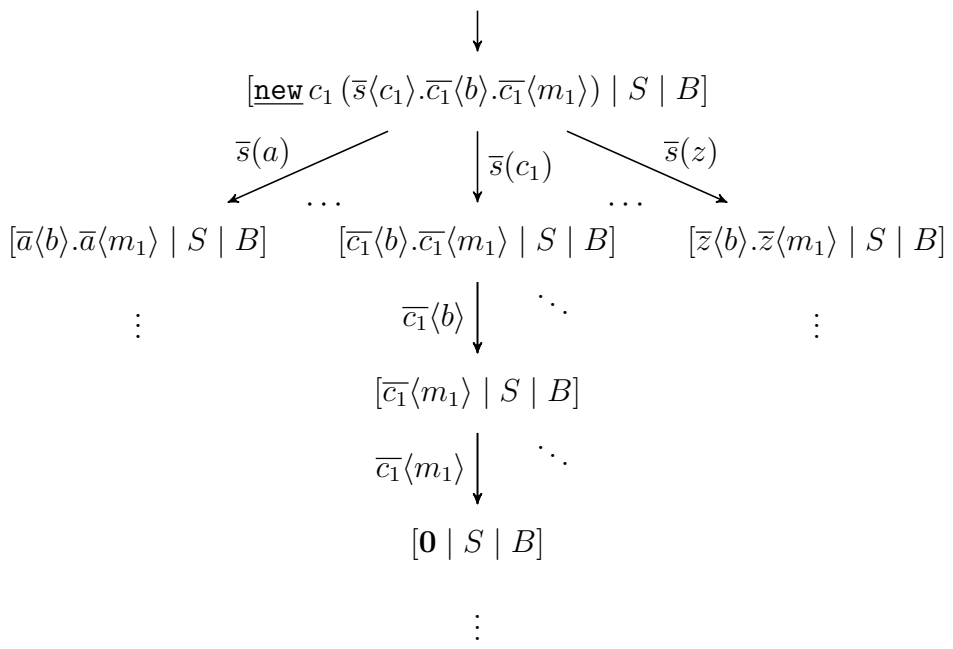

Figure 2: An extract of the early transition system of Alice's sole behavior.

holds. Where $\{^{y}/_{x}\}$ is the substitution of the name x with the name y in the process P. Since the application of the substitution is within the trace semantics, we have not obtained a compositional definition of the behavior of an input process. However, at least for every fresh name received by an input process the calculation of the traces is compositional. This stems from the equality of a transposition and a substitution of free names and the equality of the trace semantics by applying a transposition of names to the process within the semantics or to the semantics itself. That is, for a process P and a transposition σ the equality $\mathcal{T}([P\sigma]) = \mathcal{T}([P])\sigma$ holds. Furthermore, we show that the parallel composition is compositional for processes without any occurrence of restriction operators inside. The proof exploit a Gödel numbering of the traces and a new inductively defined trace calculation of the parallel composition [1]. To show that the limitation to restriction free processes for the compositionality of the parallel composition is still feasible, we extend Milner's structural congruence, such that *every* restriction operator can be moved right at the front of the process. That is, we add rules still preserving the congruence properties to extend the scope of a restriction over prefix and choice operators. We show that the limitation to restriction free processes within the compositionality of the parallel composition is not harmful, since we offer an extended standard form, to which every process can be transformed such that the resulting process is still structural congruent to the original one. A process in extended standard form has *all* restriction operators right at the front of the process. The proof of transforming a process in extended standard form is constructive and since it does not increase or reduce the number of operators, the transformation does not incur a large blowup. Furthermore, we show that the restriction operator is most likely compositional, but the proof of the one set-inclusion is not yet completely finished.

Additionally, we sketch an algorithmic idea for calculating the trace semantics for a finite set of names provided by the environment. This approach takes advantage of the

compositional aspects of the semantics.

Moreover, we show that the trace semantics preserves some algebraic laws and useful properties. For instance, we show that the prefix operator distributes over the choice operator, that is, $\mathcal{T}([\pi.M_1 + \pi.M_2]) = \mathcal{T}([\pi.\,(M_1 + M_2)])$, and state that $M_1 \sqsubseteq_{\mathcal{T}} M_2$ and $M_3 \sqsubseteq_{\mathcal{T}} M_4$ implies $M_1 + M_3 \sqsubseteq_{\mathcal{T}} M_2 + M_4$ for an arbitrary prefix π and sums $M_1, \ldots, M_4$. Furthermore, we prove that $Q \sqsubseteq_{\mathcal{T}} P$ is equivalent to $\overline{a}\langle x\rangle.Q \sqsubseteq_{\mathcal{T}} \overline{a}\langle x\rangle.P$ for all processes P, Q and names a, x.

Finally, we connect our new approach to the well-known concepts of weak and strong simulations. Thus, we prove that a weak as well as a strong simulation imply the trace refinement of every element of the simulation. That is, if a process Q simulates P (strongly or weakly, respectively), then the process P also refines Q. Furthermore, the strong or weak bisimilarity of processes implies their trace equivalence.

References

[1] Gieseking, M. Refinement of π-calculus processes. Master's thesis, Carl von Ossietzky University of Oldenburg, 2014.

[2] Hoare, C. A. R. Communicating sequential processes. *Communications of the Association for Computing Machinery 21*, 8 (1978), 666–677.

[3] Hoare, C. A. R. Why ever CSP? *Electronic Notes in Theoretical Computer Science 162* (2006), 209–215.

[4] Milner, R., Parrow, J., and Walker, D. A calculus of mobile processes, parts I and II. *Information and Computation 100*, 1 (1992), 1–77.

[5] Pistore, M., and Sangiorgi, D. A partition refinement algorithm for the π-calculus. In *Computer Aided Verification* (1996), Springer, pp. 38–49.

[6] Roscoe, A. W. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.

[7] Sangiorgi, D., and Walker, D. *The π-Calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.