Theoretical Computer Science 897 (2022) 114–134

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

A thesis for interaction

Yuxi Fu

BASICS, Shanghai Jiao Tong University, China

ARTICLE INFO

Article history:
Received 11 April 2020
Received in revised form 29 September 2021
Accepted 1 October 2021
Available online 7 October 2021
Communicated by Rob van Glabbeek

Keywords:
Process calculus
Theory of interaction
Bisimulation
Expressiveness

ABSTRACT

A thesis for interaction is proposed as a foundation for theory of interaction. It formulates the minimal content computability and the maximal channel computability of all models of interaction by fixing a minimal model and a maximal model. It is shown that the interactive extensions of the recursive function model, the Turing machine model and the λ-calculus model all fall within the realm of the thesis. A major technical contribution is the design of interpreters in the maximal model for the models that fit in the thesis.

© 2021 Published by Elsevier B.V.

1. Introduction

"The theory of computation has traditionally been studied almost entirely in the abstract, as a topic in pure mathematics. This is to miss the point of it. Computers are physical objects, and computations are physical processes. What computers can or cannot compute is determined by the laws of physics alone, and not by pure mathematics." — David Deutsch [7]

Physical implementability is the very foundation of computer science, the Church-Turing Thesis states that the functions implementable by physical devices are precisely those definable by the mathematical models of computation [23,6,56,30]. The thesis has two sides of a story. One is about definability, pointing out that the mathematical models offer a formulation of the minimal capacity of computing devices. Digital computers built with von Neumann structure are physical devices that enjoy the minimal computing power. The other is to do with unsolvability, postulating that the mathematical models characterize the maximal power of computing devices in all shapes and forms. There has been an increasing interest in the mathematical model of computation based on quantum mechanics [41]. Leaving the implementation issue aside, it has been shown that such a model is equivalent to the Turing machine model as far as computability is concerned. All experiments indicate that the world is quantum mechanical. If we believe what physicists believe, the unsolvability theory based on the Church-Turing Thesis is a consequence of the laws of physics. Many other mathematical models of computation have been studied. The probabilistic Turing machine model [52] for example, believed by many to give a broader picture of efficient computation [17], is equivalent to Turing machine model in computability. Physical implementability emphasizes an important aspect of computation, the observability. For a computing device to be useful, it should be able to fetch input data from some medium and make the result of computation deliverable. In other words both inputs and outputs ought to be observable. The requirement on observability rules out for example models that deal with general functions on the field of real numbers.

E-mail address: fu-yx@cs.sjtu.edu.cn.

https://doi.org/10.1016/j.tcs.2021.10.001
0304-3975/© 2021 Published by Elsevier B.V.

Physical implementability remains a fundamental constraint if one turns from models of computation to models of interaction. In a concurrent system processes communicate through channels; and channels are of physical nature. A frequently asked question about interaction models is if they are more powerful than computation models [24,31,57,32]. What is the expressiveness of distributed computing? How about Internet computing? For closed systems of interactive processes the Church-Turing Thesis already provides the answer to these questions, even though our everyday encounter with modern computing technology prompts us to think otherwise. The point is that a model is meant to define closed systems, and to a closed system composed of interactive physical devices the Church-Turing Thesis applies. If one is interested in open systems, there is a standard approach using oracles. Human interventions in Internet computing for example are best explained as interactions with oracles. Let's see an example. Is there a process that admits the following deterministic interactive behaviour

$$O \underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{i_1 \text{ times}} \xrightarrow{b} \underbrace{\xrightarrow{a} \ldots \xrightarrow{a}}_{i_2 \text{ times}} \xrightarrow{b} \ldots \tag{1}$$

such that $i_1 < i_2 < \ldots$ and $\{i_1, i_2, \ldots\}$ is the set of the Gödel indices of the total recursive functions? For a closed system that is immune from any external force (1) is an impossibility for otherwise there would be a process that answers deterministically if a program computes a total function. We can of course think of O as an oracle and study the consequence of admitting it into our system. From the point of view of the Church-Turing Thesis there is nothing new about interaction models. What is new is that processes have access to *external* or *global* channels. They can choose to communicate, or not to communicate at a particular channel in the middle of computation. They can also send the identifier of a channel to other processes to create new computational topology dynamically. To what extent are manipulations on channels possible? It is this aspect of interaction that has not been under scrutiny in the light of the Church-Turing Thesis. But once we have raised the issue, we immediately know the answer. Because the computational behaviour of a process is constrained by the Church-Turing Thesis, a process can only make reference to channels in a computable manner. This strongly suggests, according to the discussion in the previous paragraph, to define a mathematical model that formalizes the maximal capacity of channel manipulation.

Channels are meant to be vehicles for messages. The Church-Turing Thesis implies that the messages produced by processes are all encodable by natural numbers, no matter how the messages may look like. Once again some kind of maximality emerges. There are two points that need be discussed here. One is about the rational behind the design decision to take as atomic a communication of a message whose size is unbounded. This is adopted by the value passing calculi of Hoare [29] and Milner [33] and the higher order process calculi of Thomsen [54,55] and Sangiorgi [49,50]. But isn't it more natural to take the token-by-token communications as atomic? In the interactive machine example, is it better to let the content of a communication be just a symbol from an alphabet? One needs be careful about the answer. In an interleaving semantics there is no guarantee that a piece of information can reach its destination in integrity if it is broken down into bounded size packages. In a concurrent scenario different parts of a message may be received by different processes. In practice there are ways to solve this problem. In this paper we take that a complete piece of information is transmitted in an atomic interaction. This is to guarantee that our models are complete by the criterion of this paper [14]. The second point is concerned with dynamic creation of private channels. Since we cannot restrict the number of global channels a process needs to make use of, it is again at the right level of abstraction to assume the availability of a potentially infinite number of global channels. How about private channels? Do we need to assume that there is an infinite number of private channels? In Section 5 we will give an interesting interpretation that will shed more light on this issue.

Physical implementability may tell us a lot more about models of interaction. What we have discussed in the preceding paragraphs are those aspects most important to the main theme of the paper. We shall propose a thesis for interaction that plays a similar role for theory of interaction as Church-Turing Thesis plays for theory of computation. We shall prove that the fundamental models of computation can all be lifted to models of interaction that satisfy the thesis. After fifty years of concurrency theory [45], our understanding of concurrency theory has reached a stage where new results are built from known results and the whole theory is based upon a firm foundation. The thesis to be proposed aims to provide one such foundation. At the technical level the contribution of this paper includes the proposal of a maximal interaction model and the proofs that the models of this paper are all submodels of the maximal model. The proofs of the submodel relationship make heavy use of the universal process technique, and the proofs of the negative results rely on a generalization of the technique proposed in [14].

The work of this paper derives inspiration from a number of previous work. The barbed bisimulation [36] pioneers a model independent approach to concurrency theory. The branching bisimulation [21,22] offers a powerful tool in the model independent scenario. The theory of expressiveness is an on-going topic for research [18]. Studies in this direction [39,37,42,25,26,16] help reach a consensus on criteria for comparing process models, which eventually lead to a theory of interaction [14]. The latter is a precursor to the present work.

In concurrency theory many process models have been proposed and many process combinators have been studied [38]. The concurrent composition operator "|" for example has several variants. In majority models, say in **CCS**, "|" is an enabling operator that allows processes to communicate through shared channels. In $P \mid Q$ the processes P and Q may interact through common channels, they may also act on their own. In CSP style models there is an interface parallel operator "$|_L$".

In $P\,|_L\,Q$ the processes P, Q may synchronize at channels in L; they may not synchronize at a channel not in L. In the ambient style models [5] the composition operator "|" allows one ambient to move inside another ambient. In the theory of interaction [14] the focus is on the interaction models that have the **CCS** style binary composition operator and the standard restriction operator. An interaction is a communication between two parties via a single channel. When applying a restriction operator (p), also called localization operator, to a term T, one gets a term $(p)T$ in which the private channel p can only be seen within T. It has been shown that without the restriction operator the composition operator "|" is weaker than one expects [16]. The reason for that is that all intended communications can be interrupted by the environment. For example **CCS** without the restriction operator is not Turing complete. The theory of interaction does not claim to cover all the models studied in concurrency theory. As a sequel to [14] the models considered in this paper are of the same kind. This is why it is titled "a theory of interaction" rather than "the theory of interaction". The structural rules for our composition and restriction operators are the following.

$$\frac{T_0 \xrightarrow{\lambda} T_0'}{T_0\,|\,T_1 \xrightarrow{\lambda} T_0'\,|\,T_1} \qquad \frac{T_1 \xrightarrow{\lambda} T_1'}{T_0\,|\,T_1 \xrightarrow{\lambda} T_0\,|\,T_1'} \qquad \frac{T \xrightarrow{\lambda} T'}{(p)T \xrightarrow{\lambda} (p)T'}\ p \notin \lambda \tag{2}$$

where λ stands for an action and $p \notin \lambda$ means that p does not appear in λ. These semantic rules are common to all the models considered in this paper. We will not mention them any more.

The rest of the paper is organized as follows. Section 2 reviews a few preliminaries of the theory of interaction developed in [14]. Section 3 defines three interaction models. Section 4 postulates a thesis for interaction models and points out that the interactive Turing machine model and the interactive recursive function model satisfy the thesis. Section 5 establishes the fact that the interactive λ-calculus model and the π-calculus also satisfy the thesis. Section 6 studies the relative expressiveness between the models in terms of negative results. Section 7 points out some research directions.

This paper is a sequel to [14]. The latter paper contains nothing about the soundness of the models, which is a main topic of the present paper. The definition of the soundness and all the soundness proofs are new. Section 6 is taken from [15]. It describes a very interesting and powerful technique for proving negative results.

2. Equality and submodel

Milner-Park's bisimulation approach [33,43] to observational equivalence has been a standard in concurrency theory. Various bisimulation equalities have been studied in [35,49,28,44,10] for the π-calculus and in [54,55,49,50,58] for higher order calculi. These are weak bisimilarities without any consideration to divergence. Two refinements of Milner-Park's bisimulation have played an important role in recent studies of interaction models [14]. The first is the branching bisimulation of van Glabbeek and Weijland [21,22] that imposes the additional condition that a change-of-state internal action must be bisimulated. In addition to the bisimulation property we shall pay particular attention to divergence. There is a detailed discussion in [19] on divergence conditions with regards to branching bisimulation. The criterion we adopt for divergence sensitive equivalence was introduced by Priese [47]. To define a model independent equality for interactive objects, we assume that there is a dichotomy on the actions of every interaction model. There are internal actions and external actions. An internal transition, or a single computation step, is denoted by $\xrightarrow{\tau}$. We shall write $\Longrightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\overset{\tau}{\Longrightarrow}$ for the transitive closure of $\xrightarrow{\tau}$. The external actions are message-passing actions through global channels.

Definition 1. A relation $\mathcal{R}$ on processes is a *simulation* if at least one of the following is valid whenever $P\,\mathcal{R}\,Q$ and $P \xrightarrow{\tau} P'$:

1. $Q \Longrightarrow Q'$ and $P\,\mathcal{R}\,Q'$ and $P'\,\mathcal{R}\,Q'$ for some Q';
2. $Q \Longrightarrow Q'' \xrightarrow{\tau} Q'$ and $P\,\mathcal{R}\,Q''$ and $P'\,\mathcal{R}\,Q'$ for some Q'', Q'.

The relation $\mathcal{R}$ is a *bisimulation* if both $\mathcal{R}$ and its inverse relation $\mathcal{R}^{-1}$ are simulations. It is *codivergent* if $P\,\mathcal{R}\,Q \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} Q_i \xrightarrow{\tau} \ldots$ implies that $P \overset{\tau}{\Longrightarrow} P'\,\mathcal{R}\,Q_k$ for some $k \geq 0$ and some process P', and $\mathcal{R}^{-1}$ satisfies the same property.

Both bisimulation and codivergence are properties about computation.

The second refinement of Milner-Park's bisimulation is the barbed bisimulation of Milner and Sangiorgi [36], which provides a model independent equivalence for processes. Central to the barbed bisimulation is the requirement that equality for interactive objects should be closed under concurrent composition and preserve observability.

Definition 2. A process P is *observable at the global channel a*, notation $P\Downarrow_a$, if $P \Longrightarrow P'$ for some P' that can perform some external action at channel a.

Definition 3. A relation $\mathcal{R}$ is *extensional* if $(P\,|\,P')\,\mathcal{R}\,(Q\,|\,Q')$ whenever $P\,\mathcal{R}\,Q$ and $P'\,\mathcal{R}\,Q'$. It is *equipollent* if $P\,\mathcal{R}\,Q$ implies $P{\Downarrow_a} \Leftrightarrow Q{\Downarrow_a}$ for every global channel a.

Extensionality and equipollence are properties about interaction.

Bisimulation, codivergence, extensionality and equipollence apply to all models considered in this paper. This is because they do not refer to any particularity of external actions. Internal actions are basically the same in all models [13] and are therefore model independent.

2.1. Process equality

Putting together the conditions introduced in Definition 1 and Definition 3, we get the process equality introduced in [14]. In the rest of the paper, $\mathbb{M}, \mathbb{N}$ stand for any models considered in this paper. A process definable in $\mathbb{M}$ is often called an $\mathbb{M}$-process.

Definition 4. The *absolute equality* $=_{\mathbb{M}}$ on $\mathbb{M}$-processes is the largest reflexive, extensional, equipollent, codivergent bisimulation.

It is a standard exercise to show that the largest reflexive, extensional, equipollent, codivergent bisimulation exists [14], hence the well-definedness of $=_{\mathbb{M}}$. We need the reflexivity condition in the existence proof. Without the reflexivity the extensional closure of the binary relation $\{(!\tau\,|\,!a, !\tau\,|\,!\overline{a}) \mid a \in \mathcal{C}_g\}$ would be an equipollent, codivergent bisimulation. We shall often omit the subscript in $=_{\mathbb{M}}$. The equality $=_{\mathbb{M}}$, introduced in [14], is based on previous studies on the relative expressiveness of **CCS** variants and π variants [16]. The effectiveness of the equality $=_{\mathbb{M}}$ has been demonstrated by the proof of a number of general results. See [14] for a systematic study. In [19,20,9] the authors studied bisimulation equality with explicit divergence, which coincides with the absolute equality.

Using the absolute equality we can formally distinguish two kinds of internal action. An internal action $P \xrightarrow{\tau} P'$ is a one-step *deterministic* computation, notation $P \to P'$, if $P' = P$. It is a one-step *nondeterministic* computation, notation $P \xrightarrow{\iota} P'$, if $P' \neq P$. A fundamental property about internal actions is that $P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_k = P_0$ implies $P_0 \to P_1 \to \ldots \to P_k$ [14]. This property implies that if $Q_0 = P_0$ and $Q_0 \xrightarrow{\lambda} Q'$ is bisimulated by the sequence $P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_k \xrightarrow{\lambda} P'$, then $P_0 \to P_1 \to \ldots \to P_k$.

Let's see some examples definable in the **CCS** model [33]. The recursion term $\mu X.T$ is defined by the following transition rule

$$\frac{T\{\mu X.T/X\} \xrightarrow{\lambda} T'}{\mu X.T \xrightarrow{\lambda} T'}.$$

The first example shows the difference between our equality and the weak congruence [33]. One has $a.(A+\tau.B) \neq_{\mathbf{CCS}} a.(A+\tau.B)+a.B$ even though the two processes are weakly congruent. The reason is that $a.(A+\tau.B)+a.B \xrightarrow{a} B$ is not bisimulated by $a.(A+\tau.B) \xrightarrow{a} A+\tau.B \xrightarrow{\tau} B$ since $A+\tau.B \neq_{\mathbf{CCS}} B$. The second example exhibits the subtlety of codivergence. Let Q be $\mu X.(\tau.(a\,|\,X)+\tau+\tau.X)$ and P be $\mu X.(\tau.(a\,|\,X)+\tau)$, definable in **CCS**. We remark that $Q \xrightarrow{\tau} Q$ and $P' \neq_{\mathbf{CCS}} P$ whenever $P \stackrel{\tau}{\Longrightarrow} P'$. The process Q can engage in an infinite number of *deterministic* computation steps whereas P cannot do any *deterministic* computation. There is no codivergent equipollent bisimulation $\mathcal{R}$ such that $P\mathcal{R}Q$. The processes P, Q are different in the same way a terminating computation is different from a nonterminating computation. The process P can engage in an infinite sequence of nondeterministic computations. By the bisimulation property every process equal to P must bisimulate this infinite sequence. However the bisimulation property says nothing about infinite sequences of deterministic computation. The codivergence property takes care of that.

2.2. Submodel relationship

The equality for computational objects (processes, programs) of a model is the most fundamental relationship in computer science. Such equality is unique from a mathematical logic point of view. Another fundamental relationship in computer science is the relative expressiveness between models. The simplest such relationship is the submodel relationship. Intuitively we regard $\mathbb{M}$ as a submodel of $\mathbb{N}$ if for every $\mathbb{M}$-process P there is some $\mathbb{N}$-process Q such that Q is equal to P from the viewpoint of a bisimulation defined between $\mathbb{M}$-processes and $\mathbb{N}$-processes; and moreover the equality $=_{\mathbb{M}}$ should be a subrelation of the equality $=_{\mathbb{N}}$ under this interpretation. The reflexivity condition of Definition 4, which is a relation on one model, is generalized to conditions on processes from two different models.

Definition 5. A binary relation $\propto$ from $\mathbb{M}$-processes to $\mathbb{N}$-processes is *total* if for every $\mathbb{M}$-process P there is an $\mathbb{N}$-process Q such that $P \propto Q$. It is *sound* if $Q =_{\mathbb{N}} Q'$ whenever $P =_{\mathbb{M}} P'$ and $P \propto Q$ and $P' \propto Q'$.

If $P \propto Q$ for a subbisimilarity $\propto$, Q is called an interpretation of P. Totality says that every $\mathbb{M}$-process gets interpreted. The soundness condition forces an interpretation to be syntax independent. In the largest subbisimilarity $\propto ; =_{\mathbb{N}}$, which is the composition of $\propto$ with $=_{\mathbb{N}}$, if Q is an interpretation of P and $P' = P$ then Q is an interpretation of P' as well. Both totality and soundness are preliminary.

Definition 6. A binary relation $\propto$ from $\mathbb{M}$-processes to $\mathbb{N}$-processes is a *subbisimilarity* if it is a total, sound, extensional, equipollent, codivergent bisimulation.

Every subbisimilarity is *fully abstract*. This is the next lemma.

Lemma 2.1. *If $P \propto Q$ and $P' \propto Q'$, then $P =_{\mathbb{M}} P'$ if and only if $Q =_{\mathbb{N}} Q'$.*

Proof. The equality $Q =_{\mathbb{N}} Q'$ implies the equality $P =_{\mathbb{M}} P'$ since the composition $\propto ; =_{\mathbb{N}} ; \propto^{-1}$ is a subbisimilarity. □

We write $\mathbb{M} \sqsubseteq \mathbb{N}$ if there is a subbisimilarity from $\mathbb{M}$ to $\mathbb{N}$. The notation $\mathbb{M} \sqsubseteq \mathbb{N}$ is the formal statement that $\mathbb{M}$ is a submodel of $\mathbb{N}$. For example it is easy to see that **CCS** $\sqsubseteq \pi$. We write $\mathbb{M} \sqsubset \mathbb{N}$ if $\mathbb{M} \sqsubseteq \mathbb{N}$ and $\mathbb{N} \not\sqsubseteq \mathbb{M}$. In [14] the subbisimilarity relationship has been used to classify the relative expressiveness of several well-known models. For the relationship between the expressiveness criterion defined by the subbisimilarity and other criteria proposed in literature [42, 26], confer [14].

3. From computation models to interaction models

The interaction models based on Turing machines [56], recursive functions [48,53], and the λ-calculus [2] are defined in this section. The basic idea is to turn the input and output meta operations of the computation models into actions admitted within models. These actions are interactions between processes that pass messages through channels. To enable interactions the concurrent composition operator is admitted; and to disable interactions the scope operator is introduced. We fix the following syntactical classes for all our models.

- $\mathcal{C}_g$ is the set of global channels, ranged over by a, b, c, d, e, f.
- $\mathcal{C}_p$ is the set of private channels, ranged over by l, m, n, o, p, q.
- $\mathcal{C}_v$ is the set of channel variables, ranged over by u, v, w, x, y, z.

We write for example $\widetilde{p}$ for a finite sequence of private channels. The set $\mathcal{C}_g \cup \mathcal{C}_p \cup \mathcal{C}_v$ will be ranged over by μ, ν, and the set $\mathcal{C}_g \cup \mathcal{C}_p$ by α, β. It is worth pointing out that in [14] the private channels and the global channels are syntactically the same, following the standard practice in process algebra [33,35,51]. Consequently the extensionality condition is supposed to be closed under the scope operator. In the present paper this is unnecessary since the private channels are syntactically distinct from the global channels.

In the rest of the paper we shall write λ and its decorated versions for both internal and external actions of a model. Throughout the paper **N** denotes the set of natural numbers, and i, j, k and their decorated versions range over **N**.

3.1. From Turing machines to interactive machines

The simplest interaction model is based on Turing machines. A number of researchers have studied interactive versions of the Turing machines and their expressiveness, see the papers [24,31,3] and the references therein. Suppose every *interactive Turing machine*, ITM for short, has an input tape, an output tape and a work tape. It has a finite set of states, and a finite set of transition rules. We assume that all ITMs use the same alphabet $\{0, 1, \Box\}$. A configuration is a 7-tuple $(p, s_0, s_1, s_2, h_0, h_1, h_2)$, where p is the state, $s_0, s_1, s_2 \in \{0, 1\}^*$ are the strings written on the input tape, the work tape and the output tape, h_0, h_1, h_2 are the positions of the readers on the input tape, the work tape and the output tape. An ITM is defined by a finite transition function like a classic Turing machine. Each entry of the transition table can be seen as a rule

$$(p, e_0, e_1, e_2) \to (p', e_1', e_2', H_0, H_1, H_2).$$

It says that if the machine is in state p and the three readers are reading the cells containing the tokens e_0, e_1, e_2 respectively, then the machine overwrites e_1, e_2 with e_1', e_2' respectively, move the readers according to H_0, H_1, H_2, and turns into state p'. If the machine is in the configuration M to which one transition rule applies, then it turns to the unique configuration M' obtained from M by the rule. This one step computation is denoted by $M \xrightarrow{\tau} M'$. An ITM differs from a classic Turing machine in that they can interact with each other. An ITM is connected to a finite number of global and/or private channels. Different ITMs are connected through channels. For each channel α an ITM is connected to, the ITM contains finitely many input transitions of the form $p \xrightarrow{\alpha} p'$ and/or finitely many output transitions of the form $p \xrightarrow{\overline{\alpha}} p'$. An input transition may induce for example transition $M \xrightarrow{\alpha(s)} M'$, which says that the ITM in configuration M can input the string

s at channel α and write s on the input tape and the configuration turns into M'. An output transition may induce for example transition $M \xrightarrow{\overline{\alpha}(s)} M'$, which says that the ITM in configuration M can output s through channel α, where s is the string on the output tape.

A *machine system* M is either a machine configuration, or is a concurrent composition of the form $M' \mid M''$, or is of the form $(p)M'$. In $(p)M'$ the private channel p is *closed*. A private channel is *open* if it is not closed. The operational semantics of the machine systems is completed by the following interaction rules

$$\frac{M_0 \xrightarrow{\alpha(s)} M_0' \quad M_1 \xrightarrow{\overline{\alpha}(s)} M_1'}{M_0 \mid M_1 \xrightarrow{\tau} M_0 \mid M_1'} \qquad \frac{M_0 \xrightarrow{\overline{\alpha}(s)} M_0' \quad M_1 \xrightarrow{\alpha(s)} M_1'}{M_0 \mid M_1 \xrightarrow{\tau} M_0 \mid M_1'} . \tag{3}$$

The external actions $\alpha(s)$ and $\overline{\alpha}(s)$ are complementary, and they cooperate to achieve an interaction. Every interaction model studied in this paper has interaction rules like the ones in (3) with its own complementary external actions. We shall not mention these rules either when defining a new model.

We denote this interaction model by $\mathbb{T}$. A $\mathbb{T}$-process is a machine system in which all private channels are closed. It is easy to see that every $\mathbb{T}$-process is equal to a $\mathbb{T}$-process of the form

$$(\widetilde{p})(M_0 \mid M_1 \mid \ldots \mid M_k), \tag{4}$$

where $M_0, M_1, \ldots, M_k$ are interactive Turing machines that contain no occurrences of the concurrent composition operator. From the viewpoint of implementation this is a fine property not enjoyed by the other models of this paper.

3.2. From recursive functions to the value-passing calculus

The value-passing calculus [27,28,12], denoted by $\mathbb{R}$ in this paper, is the interactive recursive function model. The contents of communications in $\mathbb{R}$ are numbers. Formally we use Presburger Arithmetic [46] to reason about numbers. Presburger Arithmetic is the first order theory of arithmetic with three functional constructors (the nullary function “0”, the unary function successor “$+1$”, and the binary function addition “$+$”) and two relations (“$=$” and “$<$”). We will write r, s, t for Presburger terms and x, y, z for natural number variables. We will write $\vdash \varphi$ to mean that the Presburger formula φ is provable. It is well known that the provability of the first order formula in Presburger Arithmetic is decidable [46]. The syntax of $\mathbb{R}$ is given by the following grammar.

$$S, T := \mathbf{0} \mid \alpha(x).T \mid \overline{\alpha}(t).T \mid S \mid T \mid (p)T \mid [\varphi]T \mid !\alpha(x).T.$$

The nil process $\mathbf{0}$ and the composition term $T \mid T'$ are standard. A trailing $\mathbf{0}$ is omitted. A prefix term is either an input term of the form $\mu(x).T$ or an output term of the form $\overline{\mu}(t).T$. In the conditional term $[\varphi]T$, often denoted by *if* φ *then* T, the expression φ is a Presburger formula. The semantics of $\mathbb{R}$ is defined by the structural and interaction rules plus the following.

$$\frac{}{\alpha(x).T \xrightarrow{\alpha(i)} T\{i/x\}} \qquad \frac{}{\overline{\alpha}(t).T \xrightarrow{\overline{\alpha}(i)} T} \vdash t = i \qquad \frac{T \xrightarrow{\lambda} T'}{[\varphi]T \xrightarrow{\lambda} T'} \vdash \varphi$$

$$\frac{}{!\alpha(x).T \xrightarrow{\alpha(i)} T\{i/x\} \mid !\alpha(x).T} \tag{5}$$

The input guarded replication term $!\alpha(x).T$ is a typical way of introducing recursively defined processes. The replication rule (5) is standard. We shall not mention it again when defining a new model. A detailed exposure of $\mathbb{R}$ can be found in [12].

3.3. From the λ-calculus to the higher order process calculus

Church's λ-calculus [2] is a prototypical higher order model for functional computation. It is of higher order in that an input to a λ-term is a λ-term and the value of a λ-term evaluation is a λ-term. Its syntax is defined by the grammar

$$M := x \mid \lambda x.M \mid MM'.$$

A λ-term is either a variable x, or an abstraction term $\lambda x.M$, or an application term MM'. The variable x in $\lambda x.M$ is bound. A variable is free if it is not bound. A term is closed if all variables appearing in it are bound. The operational semantics most relevant in the present setting is that of the lazy λ-calculus [1], defined by the following two rules:

$$(\lambda x.M)N \rightarrow M\{N/x\},$$
$$MN \rightarrow M'N, \ \text{ if } M \rightarrow M'.$$

The equivalence closure of the one step lazy reduction $\rightarrow$ is the β-conversion $=_\beta$.

We intend to promote the lazy λ-calculus to an interaction model in which the content of a communication is a binary abstraction [49,58]. The reason that we confine to binary abstractions is that they are the natural generalization of λ-terms. Intuitively in a binary abstraction the first parameter can be instantiated by a channel through which a 'λ-term' can be imported and the second parameter can be instantiated by a channel through which the result of evaluation can be delivered. Under this interpretation a λ variable can be seen as an abstraction variable. This will become clear in Section 3.4 where λ-terms get interpreted as binary abstraction. Let $\mathcal{V}$ be the set of abstraction variables, ranged over by U, V, W, X, Y, Z. The set of $\mathbb{L}$-*terms* is generated by the following grammar:

$$S, T := \mathbf{0} \mid \mu(X).T \mid \overline{\mu}[A].T \mid S \mid T \mid (p)T \mid A(\mu,\nu) \mid !\mu(X).T,$$
$$A := X \mid (x,y)T.$$

An abstraction is either an *abstraction variable* or of the form $(x,y)T$. In $(x,y)T$ the channel variables x, y are *bound*. A channel variable is *free* if it is not bound. If y does not appear in T, we abbreviate $(x,y)T$ to $(x)T$. If $A=(x,y)T$, then the instantiation of A at μ,ν is $A(\mu,\nu)=T\{\mu/x,\nu/y\}$. If $A=(x)T$, then the instantiation $A(\mu)$ of A at μ is $A(\mu)=T\{\mu/x\}$. We will write $A_{\mu,\nu}$ or even $A_{\mu\nu}$ for $A(\mu,\nu)$ and A_μ for $A(\mu)$ to improve readability. The set of abstractions is ranged over by A, B, C, D. The abstraction variable X in $\mu(X).T$ is *bound*. A (abstraction) variable is free if it is not bound. We shall abbreviate $\mu(X).T$ to $\mu.T$ if X does not appear in T. Similarly we shall abbreviate $\overline{\mu}[\mathbf{0}].T$ to $\overline{\mu}.T$.

In the standard semantics for higher order models, there are transitions that look like

$$(p)(S \mid \overline{a}[\ldots\overline{c}[\_].\overline{p}[\_]\ldots]) \mid a(X).T \xrightarrow{\tau} (p)(S \mid T\{\ldots\overline{c}[\_].\overline{p}[\_]\ldots/X\}), \tag{6}$$

where the global channel c does not appear in T. The scope operator (p) is extended to cover T after the interaction, which does not really make sense if one thinks of p as a local physical line. Another problem is that having landed in T the component $\overline{c}[\_].\overline{p}[\_]\ldots$ can communicate to a process at channel c and can secretly pass information to its original host through the private channel p. The party that has received $\overline{c}[\_].\overline{p}[\_]\ldots$, which is T, has absolutely no control over either of the actions. From the point of view of T the term $\ldots\overline{c}[\_].\overline{p}[\_]\ldots$ is more like a spy than a useful piece of information. In retrospect higher order communications should be defined in such a way to avoid situations as described in (6) from ever happening in the first place. A receiving party should have complete control over the messages it has obtained from another party. For this purpose we impose on $\mathbb{L}$ the following conditions.

- There are no free channel variables in any abstraction that appears in an output prefix.
- There are no open private channels in any abstraction that appears in an output prefix.
- There are no global channels in any abstraction that appears in an output prefix.

In $\mathbb{L}$ an abstraction generalizes a closed λ-term.

Using the action set $\{\alpha(A), \overline{\alpha}(A) \mid \alpha \in \mathcal{C}_g \cup \mathcal{C}_p\} \cup \{\tau\}$, the operational semantics of $\mathbb{L}$ is defined by

$$\frac{}{\alpha(X).T \xrightarrow{\alpha(A)} T\{A/X\}} \qquad \frac{}{\overline{\alpha}[A].T \xrightarrow{\overline{\alpha}(A)} T}.$$

An $\mathbb{L}$-*process* is an $\mathbb{L}$-term in which all channel variables and abstraction variables are bound and all private channels are closed. We write L, M, N, O, P, Q for processes.

3.4. $\mathbb{L}$ as the λ-calculus for concurrency

Given an injective function from the set of λ variables to $\mathcal{V}$, a λ-term M is interpreted as an *abstraction* by structural induction, which is a more straightforward interpretation than the one in the π-calculus [34].

$$[\![x]\!] = X,$$
$$[\![\lambda x.M]\!] = (uv)u(X).\overline{v}[[\![M]\!]],$$
$$[\![MN]\!] = (uv)(mq)([\![M]\!]_{m,q} \mid \overline{m}[[\![N]\!]].q(Z).Z_{u,v}).$$

The process $[\![M]\!]_{a,b}$ is a "function" that inputs a "λ-term" at channel a and outputs the result "λ-term" at channel b. The structural aspect of the interpretation is enforced by the following simple lemma.

Lemma 3.1. $[\![M\{N/X\}]\!] \equiv [\![M]\!]\{[\![N]\!]/X\}$.

The interpretation is also semantically correct, guaranteed by the following facts. Notice that since we do not define reductions for abstractions, we can only talk about reduction of $[\![M]\!]$ in terms of that of $[\![M]\!]_{a,b}$ for arbitrarily chosen a, b. As pointed out already, $[\![M]\!]_{a,b}$ abbreviates $[\![M]\!](a,b)$.

Lemma 3.2. *Suppose $P \mid \mathbf{0} \equiv P \equiv \mathbf{0} \mid P$ for all P, and $(p)P \equiv P$ for all P. For closed λ-terms M, N, $M \to N$ if and only if $[\![M]\!]_{a,b} \to\to\equiv [\![N]\!]_{a,b}$.*

Proof. Let a closed λ-term M be for example of the form $(\lambda x.L)M_1M_2$, where the application is associative to the left. Now

$$\begin{aligned}
[\![M]\!]_{a,b} &\equiv (m_2q_2)\big(\ldots(m_1q_1)([\![\lambda x.L]\!]_{m_1,q_1} \mid \overline{m_1}[[\![M_1]\!]].q_1(Z).Z_{m_2,q_2})\ldots\big) \\
&\to (m_2q_2)\big(\ldots(m_1q_1)(\overline{q_1}[[\![L]\!]]\{[\![M_1]\!]/X\} \mid q_1(Z).Z_{m_2,q_2})\ldots\big) \\
&\equiv (m_2q_2)\big(\ldots(m_1q_1)(\overline{q_1}[[\![L\{M_1/x\}]\!]] \mid q_1(Z).Z_{m_2,q_2})\ldots\big) \\
&\to (m_2q_2)\big([\![L\{M_1/x\}]\!]_{m_2,q_2} \mid \overline{m_2}[[\![M_2]\!]].q_2(Z).Z_{m_3,q_3}\big) \\
&\equiv [\![L\{M_1/x\}M_2]\!]_{a,b},
\end{aligned}$$

where the middle equivalence $\equiv$ is due to Lemma 3.1. □

Lemma 3.3. *For closed λ-terms M, N, $M =_\beta N$ implies $[\![M]\!]_{a,b} =_{\mathbb{L}} [\![N]\!]_{a,b}$.*

Proof. In view of Lemma 3.2 this is the Church-Rosser property [2]. □

It is well-known [2] that there is a λ-term encoding $\ulcorner 0 \urcorner, \ulcorner 1 \urcorner, \ulcorner 2 \urcorner, \ldots, \ulcorner i \urcorner, \ldots$ of the natural numbers and an encoding $\ulcorner \_ \urcorner$ of the recursive functions such that $\ulcorner \mathsf{f} \urcorner \ulcorner i \urcorner \to^* \ulcorner \mathsf{f}(i) \urcorner$ if $\mathsf{f}(i)$ is defined, and that $\ulcorner \mathsf{f} \urcorner \ulcorner i \urcorner \to\to \ldots$ diverges if $\mathsf{f}(i)$ is undefined. Given global channels a, b we define the $\mathbb{L}$-processes

$$\overline{a}(i) \equiv \overline{a}[[\![\ulcorner i \urcorner]\!]], \tag{7}$$

$$\lambda_{a,b}(\mathsf{f}) \equiv [\![\ulcorner \mathsf{f} \urcorner]\!]_{a,b}. \tag{8}$$

It is straightforward to verify that if $\mathsf{f}(i) = j$ then

$$\lambda_{a,b}(\mathsf{f}) \mid \overline{a}(i) \xrightarrow{\iota} \overline{b}(j) \tag{9}$$

and if $\mathsf{f}(i)$ is undefined then

$$\lambda_{a,b}(\mathsf{f}) \mid \overline{a}(i) \xrightarrow{\iota}\to\to \ldots. \tag{10}$$

4. Thesis of interaction

The concurrent composition operator is an enabling combinator that allows processes to interact on shared channels. No extra computational meaning should be assigned to this operator. The composition operator is a spatial operator. It would be much less useful without any hierarchical structure in space. Private channels together with the scope operator are introduced to create space hierarchy. In addition to these two operators we need a sequential operator, a conditional, and recursion to define Turing computation. As we have said in Section 1, the models we consider in this paper all have the composition operator and the localization operator. All other operators add only computational power to the models. As a consequence all references to channels are computational. If we could define an interaction model in which the computational aspect of channel reference is formalized within the model, it is likely that we get a universal model. This will be the model $\mathbb{V}$ defined in Section 4.2.

4.1. Initial model

The starting point to formulate a thesis for interaction is to turn the computable functions into an interaction model. This model is called *computability model* and is denoted by $\mathbb{C}$ [14]. The $\mathbb{C}$-processes are generated by the grammar

$$P := \mathbf{0} \mid \Omega \mid F_a^b(\mathsf{f}(x)) \mid \overline{a}(i) \mid P \mid P.$$

The process Ω can only diverge. The functional process $F_a^b(\mathsf{f}(x))$ inputs a number, say i, at channel a, and outputs $\mathsf{f}(i)$ at channel b if the *computable* function f is defined at i. The output process $\overline{a}(i)$ simply outputs i at channel a. The semantics is defined by the following rules, in which $\mathsf{f}(i)\uparrow$ means that f is undefined on i.

$$\frac{}{\Omega \xrightarrow{\tau} \Omega} \qquad \frac{}{F_a^b(\mathsf{f}(x)) \xrightarrow{a(i)} \overline{b}(j)}\ \text{if } \mathsf{f}(i) = j$$

$$\frac{}{\overline{a}(i) \xrightarrow{\overline{a}(i)} \mathbf{0}} \qquad \frac{}{F_a^b(\mathsf{f}(x)) \xrightarrow{a(i)} \Omega}\ \text{if } \mathsf{f}(i)\uparrow$$

Definition 4 can be readily applied to the $\mathbb{C}$-processes. In $\mathbb{C}$ the absolute equality has a simple characterization. Let the structural congruence $\equiv_{\mathbb{C}}$ be the least equivalence and congruence relation that satisfies the following equalities: (i) $\mathbf{0} \mid P \equiv_{\mathbb{C}} P$, (ii) $P \mid Q \equiv_{\mathbb{C}} Q \mid P$, (iii) $O \mid (P \mid Q) \equiv_{\mathbb{C}} (O \mid P) \mid Q$, (iv) $\Omega \mid \Omega \equiv_{\mathbb{C}} \Omega$, and (v) $F_a^b(\mathsf{f}(x)) = F_a^{b'}(\mathsf{f}(x))$ if $\mathsf{f}(x)$ is nowhere defined. In [14] it is proved that the absolute equality $=_{\mathbb{C}}$ coincides with the structural congruence $\equiv_{\mathbb{C}}$. This is important in the context of this paper because it greatly simplifies the proof of soundness statements of the form $\mathbb{C} \sqsubseteq \mathbb{M}$. The simplification is due to the fact that it is easy to show that a translation is structural. We shall not prove any soundness claim in this paper.

4.2. Universal model

The model $\mathbb{C}$ introduces a minimal expressive power on the interaction models, the *content computability*. In the other direction the maximal interaction model imposes a limit, which we call *channel computability*, on the power of all interaction models. The maximal model, or universal model $\mathbb{V}$ is an extension of the value-passing calculus $\mathbb{R}$ of Hoare [29] and Milner [33]. It differs from $\mathbb{R}$ in that $\mathbb{V}$ has a *channel function* $\kappa : \mathbf{N} \rightarrow \mathcal{C}_g$ that assigns a distinct index to each global channel in a bijective way. For a number i we will write κ_i for $\kappa(i)$. The channel function is meant to capture the intuition that global channels must be accessed in a computable fashion. The function κ gives rise to an enumeration $\kappa_0, \kappa_1, \ldots$ of the global channels. When we write a global channel a for instance we mean that $a = \kappa_k$ for some index k. A process may refer to a as long as it knows a's index. The syntax of $\mathbb{V}$ is almost the same as $\mathbb{R}$, except that in $\mathbb{V}$ the prefix terms are of the form $\kappa_z(x).T$ and $\overline{\kappa_z}(t).T$. In the following definition of grammar the subscript h is either a number or a number variable.

$$S, T := \mathbf{0} \mid \kappa_h(x).T \mid \overline{\kappa_h}(t).T \mid S \mid T \mid (p)T \mid \mathit{if}\, \varphi \;\mathit{then}\; T \mid !\kappa_h(x).T.$$

If h is a number say 8, κ_h is the channel with index 8, or the 8-th channel. If h is a number variable say x, κ_h can be fixed by instantiating x by a number. The semantics of $\mathbb{V}$ are the same as that of $\mathbb{R}$. We will prove that the processes definable in $\mathbb{V}$ can simulate the channel-passing communication mechanism. We get a better understanding of the channel computability from the following abbreviation.

$$\kappa_e(x).T = (p)(\overline{p}(e) \mid p(z).\kappa_z(x).T),$$
$$\overline{\kappa_e}(t).T = (p)(\overline{p}(e) \mid p(z).\overline{\kappa_z}(t).T),$$

where p, z are fresh, and e is a number expression. Now $a(x).\overline{\kappa_{decode(x)}}(t)$ for instance decodes the number received at channel a, and if the decoded number is some k, it outputs the value of t at a channel κ_k whose index is not known to any eavesdropper. The two-leg conditional term $\mathit{if}\, \varphi \;\mathit{then}\; S \;\mathit{else}\; T$ is defined by $\mathit{if}\, \varphi \;\mathit{then}\; S \mid \mathit{if}\, \neg\varphi \;\mathit{then}\; T$. For clarity the term

$$\mathit{if}\; t = i_0 \;\mathit{then}\; T_0 \;\mathit{else}\; \mathit{if} \;\ldots\; \mathit{else}\; \mathit{if}\; t = i_k \;\mathit{then}\; T_k$$

will be written as **case** t **of** $i_0 \Rightarrow T_0;\ \ldots;\ i_k \Rightarrow T_k$ **end case**.

The absolute equality of the maximal model is faithful over the extensional equality of the computable functions. This is the characterization given by the next lemma [12].

Lemma 4.1. *The absolute equality $=_{\mathbb{V}}$ is the largest codivergent bisimulation $\mathcal{R}$ such that the following are valid whenever $P\mathcal{R}Q$ and $\lambda \neq \tau$.*

1. *If $P \xrightarrow{\lambda} P'$ then $Q \Longrightarrow Q'' \xrightarrow{\lambda} Q'$ for some Q'', Q' such that $P\mathcal{R}Q''$ and $P'\mathcal{R}Q'$.*
2. *If $Q \xrightarrow{\lambda} Q'$ then $P \Longrightarrow P'' \xrightarrow{\lambda} P'$ for some P'', P' such that $P''\mathcal{R}Q$ and $P'\mathcal{R}Q'$.*

4.3. Models inbetween

We have said enough motivations for the thesis, and we have defined the minimal model $\mathbb{C}$ and the maximal model $\mathbb{V}$. Let's spell out the thesis.

Thesis of Interaction. $\forall \mathbb{M}. \mathbb{C} \sqsubseteq \mathbb{M} \sqsubseteq \mathbb{V}$.

The formulation of the thesis depends crucially on the *model independence* of the submodel relationship, equivalently of the absolute equality. The coincidence between $=_{\mathbb{C}}$ and $\equiv_{\mathbb{C}}$ points out that the absolute equality on the initial model is completely determined by the extensional equality of the computable functions. According to Lemma 4.1 in the maximal model the absolute equality is the interactive expansion of the extensional equality of the computable functions. Thesis of Interaction forces the equality of every interaction model to be faithful over the extensional equality of computation.

In $\mathbb{C}$ all implementation details are suppressed. All deterministic computations are caused by Ω, whereas all interactions at global channels induce nondeterministic computations [14]. In $\mathbb{V}$ the computable functions are explicitly programmed, the executions of the programmes give rise to deterministic computations. When confined to internal actions, the Thesis of Interaction provides a formulation of the Church-Turing Thesis.

We call $\mathbb{M}$ *complete* if $\mathbb{C} \sqsubseteq \mathbb{M}$ and *sound* if $\mathbb{M} \sqsubseteq \mathbb{V}$. The model $\mathbb{C}$ is the starting point for the theory of definability of $\mathbb{M}$, and $\mathbb{V}$ provides the theory of unsolvability. We can now state the main result of the paper.

Theorem 4.2. $\mathbb{C} \sqsubset \mathbb{T} \sqsubset \mathbb{R} \sqsubset \mathbb{V}$ *and* $\mathbb{C} \sqsubset \mathbb{L} \sqsubseteq \mathbb{R} \sqsubset \mathbb{V}$.

Proof. We sketch the proof. It is obvious that $\mathbb{R} \sqsubseteq \mathbb{V}$. No $\mathbb{R}$-process can interpret the $\mathbb{V}$-process $a(x).\kappa_x(0)$, hence $\mathbb{V} \not\sqsubseteq \mathbb{R}$. The proof of the submodel relationship $\mathbb{T} \sqsubseteq \mathbb{R}$ is a formality. It is essentially the encoding of the Turing machines by the recursive functions. The negative result $\mathbb{R} \not\sqsubseteq \mathbb{T}$ is due to the fact that the replication process $!a(x).\overline{b}(x)$ is not equal to any communicating sequential process in the form of (4). The completeness of $\mathbb{T}$ is clear. The completeness of $\mathbb{L}$ is proved in Section 3.4, see the processes defined in (7) and (8) and the reductions in (9) and (10). The proof of $\mathbb{L} \sqsubseteq \mathbb{R}$ will be given in Section 5. Neither $\mathbb{T} \sqsubseteq \mathbb{C}$ nor $\mathbb{L} \sqsubseteq \mathbb{C}$ because $(p)(\overline{p} \mid p.\overline{b} \mid p.\overline{c})$ cannot be interpreted by any $\mathbb{C}$-process. □

5. Interpreter

This section serves two purposes. One is to introduce the important technique of constructing an interpreter of one model in another. We explain the technique by designing a $\mathbb{V}$-process that interprets $\mathbb{L}$. The other is to demonstrate how to apply this technique to establish submodel relationship. We do this by showing $\mathbb{L} \sqsubseteq \mathbb{R}$.

For the interpreter to work, there should be a way to translate an $\mathbb{L}$-process to a number, and vice versa. This is what Gödel encoding is about.

5.1. Gödel index

For each k let $\underbrace{\langle \_, \ldots, \_ \rangle}_{k} : \underbrace{\mathbf{N} \times \ldots \times \mathbf{N}}_{k} \to \mathbf{N}$ be an effective tupling function. The corresponding projection functions are denoted by $(\_)^k_0, \ldots, (\_)^k_{k-1}$. We often write z_i for the i-th projection when k is understood from context. For notational clarity we even abbreviate $((z)^k_i)^{k'}_j$ to $z_{i,j}$. For every k we define $\mathsf{r}_k, \mathsf{d}_k$ by

$$\mathsf{r}_k(z) \overset{\text{def}}{=} \begin{cases} 0, & \text{if } z = 0, \\ i, & \text{if } 1 \le i \le k \text{ and } \exists j. z = k * j + i, \end{cases}$$

$$\mathsf{d}_k(z) \overset{\text{def}}{=} \begin{cases} 0, & \text{if } z = 0, \\ j, & \text{if } z = k * j + \mathsf{r}_k(z). \end{cases}$$

The Gödel encoding is based on two bijections. The function $\varsigma : \mathcal{V} \to \mathbf{N}$ provides a countable enumeration $\{X_0, X_1, \ldots\}$ of the abstraction variables. The bijective map $\sigma : \mathcal{C}_v \cup \mathcal{C}_g \cup \mathcal{C}_p \to \mathbf{N}$ is defined by

$$\sigma(\mu) = \begin{cases} 3i, & \text{if } \mu = x_i, \\ 3i+1, & \text{if } \mu = \kappa_i, \\ 3i+2, & \text{if } \mu = p_i, \end{cases} \tag{11}$$

assuming an enumeration of the private channels $\{p_0, p_1, \ldots\}$ and an enumeration of the channel variables $\{x_0, x_1, \ldots\}$, and making use of the channel function κ. The encoding function $[\![\_]\!]_{\mathfrak{f}}$ for terms is defined as follows.

$$\begin{aligned}
[\![\mathbf{0}]\!]_{\mathfrak{f}} &\overset{\text{def}}{=} 0, \\
[\![\mu(X).T]\!]_{\mathfrak{f}} &\overset{\text{def}}{=} 5 * \langle \sigma(\mu), \varsigma(X), [\![T]\!]_{\mathfrak{f}} \rangle + 1, \\
[\![\overline{\mu}[A].T]\!]_{\mathfrak{f}} &\overset{\text{def}}{=} 5 * \langle \sigma(\mu), [\![A]\!]_{\mathfrak{f}}, [\![T]\!]_{\mathfrak{f}} \rangle + 2, \\
[\![S \mid T]\!]_{\mathfrak{f}} &\overset{\text{def}}{=} 5 * \langle [\![S]\!]_{\mathfrak{f}}, [\![T]\!]_{\mathfrak{f}} \rangle + 3, \\
[\![(p)T]\!]_{\mathfrak{f}} &\overset{\text{def}}{=} 5 * \langle \sigma(p), [\![T]\!]_{\mathfrak{f}} \rangle + 4, \\
[\![A(\mu, \nu)]\!]_{\mathfrak{f}} &\overset{\text{def}}{=} 5 * \langle \sigma(\mu), \sigma(\nu), [\![A]\!]_{\mathfrak{f}} \rangle + 5,
\end{aligned}$$

and

$$[\![A]\!]_{\mathfrak{f}} \overset{\text{def}}{=} \begin{cases} 2 * \varsigma(X), & \text{if } A = X, \\ 2 * \langle \sigma(x), \sigma(y), [\![T]\!]_{\mathfrak{f}} \rangle + 1, & \text{if } A = (x, y)T. \end{cases}$$

$$
\begin{array}{rcl}
S_n & = & (pq)!n(z).\ \mathbf{case}\ \mathsf{r}_5(z)\ \mathbf{of} \\
& & \quad 1 \Rightarrow \mathbf{case}\ \mathsf{r}_3(\mathsf{d}_5(z)_0)\ \mathbf{of}\ 1 \Rightarrow \kappa_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}(x).\overline{n}([x/\mathsf{d}_5(z)_1]\mathsf{d}_5(z)_2); \\
& & \qquad\qquad\qquad\qquad\qquad\quad 2 \Rightarrow \overline{l_i}.\overline{p}(z) \\
& & \qquad\quad \mathbf{end\ case}; \\
& & \quad 2 \Rightarrow \mathbf{case}\ \mathsf{r}_3(\mathsf{d}_5(z)_0)\ \mathbf{of}\ 1 \Rightarrow \overline{\kappa_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}}(\mathsf{d}_3(\mathsf{d}_5(z)_1)).\overline{n}(\mathsf{d}_5(z)_2); \\
& & \qquad\qquad\qquad\qquad\qquad\quad 2 \Rightarrow \overline{l_o}.\overline{q}(z) \\
& & \qquad\quad \mathbf{end\ case}; \\
& & \quad 3 \Rightarrow \overline{n}(\mathsf{d}_5(z)_0)\,|\,\overline{n}(\mathsf{d}_5(z)_1); \\
& & \quad 4 \Rightarrow m(u).\left(\overline{m}(u+3)\,|\,(o)(In(u)\,|\,Out(u)\,|\,\overline{n}([u/\mathsf{d}_5(z)_0]\mathsf{d}_5(z)_1))\right); \\
& & \quad 5 \Rightarrow \overline{n}\,([\mathsf{d}_5(z)_0/\mathsf{d}_2(\mathsf{d}_5(z)_2)_0, \mathsf{d}_5(z)_1/\mathsf{d}_2(\mathsf{d}_5(z)_2)_1]\,\mathsf{d}_2(\mathsf{d}_5(z)_2)_2) \\
& & \mathbf{end\ case}. \\
In(y) & = & p(z).\mathit{if}\ \mathsf{d}_5(z)_0 = y\ \mathit{then}\ B_i\ \mathit{else}\ F_i, \\
B_i & = & m_i(v).\mathit{if}\ v > 0\ \mathit{then}\ \overline{l'}.\overline{m_i}(v).In(y)\ \mathit{else}\ l.I(z), \\
F_i & = & m_i(v).\overline{p}z.\overline{m_i}(v{+}1).l'.m_i(v).\mathit{if}\ v > 1\ \mathit{then}\ \overline{l'}.\overline{m_i}(v{-}1).In(y)\ \mathit{else}\ \overline{l}, \\
I(z) & = & o(x).l_i.(\overline{n}([x/\mathsf{d}_5(z)_1]\mathsf{d}_5(z)_2)\,|\,In(y)). \\
Out(y) & = & q(z).\ \mathit{if}\ \mathsf{d}_5(z)_0 = y\ \mathit{then}\ B_o\ \mathit{else}\ F_o, \\
B_o & = & m_o(v).\mathit{if}\ v > 0\ \mathit{then}\ \overline{l'}.\overline{m_o}(v).Out(y)\ \mathit{else}\ l.O(z), \\
F_o & = & m_o(v).\overline{q}z.\overline{m_o}(v{+}1).l'.m_o(v).\mathit{if}\ v > 1\ \mathit{then}\ \overline{l'}.\overline{m_o}(v{-}1).Out(y)\ \mathit{else}\ \overline{l}, \\
O(z) & = & \overline{o}(\mathsf{d}_5(z)_1).l_o.(\overline{n}(\mathsf{d}_5(z)_2)\,|\,Out(y)). \\
L_i & = & l_i.\overline{l_i}.L_i, \\
L_o & = & l_o.\overline{l_o}.L_o.
\end{array}
$$

Fig. 1. Simulator S_n.

Given an $\mathbb{L}$-term T, $[\![T]\!]_{\mathfrak{f}}$ is the *Gödel index* of T, and $[\![A]\!]_{\mathfrak{f}}$ is the Gödel index of A. For the interpreter to work properly, every natural number is understood as the index of some abstraction. If k is not $[\![A]\!]_{\mathfrak{f}}$ for any abstraction A, it is deemed as an index of $(x_0, x_1)\mathbf{0}$.

5.2. Interpreter

To talk about Gödel index in $\mathbb{L}$, it suffices to explain how numbers are defined in $\mathbb{L}$. A $\mathbb{V}$-*interpreter* $\mathcal{I}^\alpha$ of $\mathbb{L}$ at α inputs a number at channel α, interprets it as the Gödel index of some $\mathbb{L}$-process, and then simulates the $\mathbb{L}$-process. The process $\mathcal{I}^\alpha$ consists of a *parser* and a *simulator* and is defined by the $\mathbb{V}$-process

$$\alpha(x).(l_i l_o m_i m_o m n o)\left(P_n(x)\,|\,S_n\,|\,\overline{m_i}(0)\,|\,\overline{m_o}(0)\,|\,\overline{m}(k_x)\,|\,L_i\,|\,L_o\right).$$

The locks L_i, L_o are necessary to prevent the simulator S_n from running into deadlock. The numbers $0, 0, k_x$ provide the initial values of three counters accessible at m_i, m_o, m respectively. The expression k_x stands for 3 plus the maximum of the indices of the private channels appearing in x, which can be easily calculated from x.

The parser $P_n(x)$ parses a number imported at channel α to see if it is a Gödel index of some $\mathbb{L}$-process. It needs to check the following:

1. Are all channel variables bound?
2. Are all private channels closed?
3. Are all abstraction variables bound?

If the input number is the Gödel index of some $\mathbb{L}$-process P, $P_n(x)$ sends the index to the simulator through channel n. The operations performed by $P_n(x)$ are arithmetical. So it is definable in $\mathbb{L}$ by the completeness of $\mathbb{L}$.

The simulator S_n defined in Fig. 1 receives a Gödel index from the parser at channel n and simulates the operations of P on-the-fly. For clarity the terms $In(y)$ and $Out(y)$ are defined recursively. They can be easily implemented in terms of the replication operator. The simulator makes use of the channel function to simulate transmissions at global channels. The simulation of an interaction at a private channel is a bit involved. Each term of the form $(q)T$ is translated to a $\mathbb{V}$-term, called a *dominion*, of the form

$$(o)\left((l'l)In(u)\,|\,(l'l)Out(u)\,|\,\overline{n}\left([u/\varsigma(q)][\![T]\!]_{\mathfrak{f}}\right)\right). \qquad (12)$$

We need to explain the notation $[u/\varsigma(q)][\![T]\!]_{\mathfrak{f}}$ in some details. Recall that the *number* $[\![T]\!]_{\mathfrak{f}}$ is defined structurally. The number $[u/\varsigma(q)][\![T]\!]_{\mathfrak{f}}$ is defined like $[\![T]\!]_{\mathfrak{f}}$, except that in the structural definition of the former the number $\varsigma(q)$ is replaced by the number u. This is clearly computable. The operation $[u/\varsigma(q)]$ maps the number $[\![T]\!]_{\mathfrak{f}}$ onto the number $[u/\varsigma(q)][\![T]\!]_{\mathfrak{f}}$.

Different dominions make use of the same symbol o for private channels with *disjoint* scopes. In other words all private channels of an $\mathbb{L}$-process are translated *syntactically* to o. Since the simulator S_n can only refer to a finite number of symbols, this is essentially the only option. The component $\overline{n}([u/\varsigma(q)][\![T]\!]_{\mathfrak{f}})$ invokes the simulator S_n recursively, which may take $[u/\varsigma(q)][\![T]\!]_{\mathfrak{f}}$ out of the dominion it initially stays. The continuation of $[u/\varsigma(q)][\![T]\!]_{\mathfrak{f}}$ typically wanders through different dominions. The question is then how to ensure that the private channel q appearing in T and any continuation of T gets interpreted correctly. The solution is to attach to every dominion a distinct number u. If a continuation of $[u/\varsigma(q)][\![T]\!]_{\mathfrak{f}}$ tries to communicate at the private channel o that is the interpretation of q, it must first of all move to the dominion where $\overline{n}([u/\varsigma(q)][\![T]\!]_{\mathfrak{f}})$ was produced in the first place, and communicate through the private channel o effective in that dominion. The movements have to be regulated so that no deadlock can occur.

We now explain the execution of S_n in some detail. After receiving the index z of an $\mathbb{L}$-process, S_n checks the type of the process.

1. If $\mathsf{r}_5(z)=1$ then the number codes up an input process. There are two subcases:
   (a) $\mathsf{r}_3(\mathsf{d}_5(z)_0)=1$. That is the process that intends to input something at the global channel $\kappa_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}$. If h is received at channel $\kappa_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}$, then S_n simulates the process with the index $[h/\mathsf{d}_5(z)_1]\mathsf{d}_5(z)_2$ obtained by replacing $\mathsf{d}_5(z)_1$ by h throughout the index $\mathsf{d}_5(z)_2$.
   (b) $\mathsf{r}_3(\mathsf{d}_5(z)_0)=2$. This is the process ready to input something at the private channel with code number $\mathsf{d}_3(\mathsf{d}_5(z)_0)$. This is reciprocal to Case 2(b).
2. If $\mathsf{r}_5(z)=2$, it is an output process. There are two cases:
   (a) $\mathsf{r}_3(\mathsf{d}_5(z)_0)=1$. In this case the process is ready to output the index of an abstraction at the global channel $\kappa_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}$. After simulating this action, S_n is invoked with the index $\mathsf{d}_5(z)_2$.
   (b) $\mathsf{r}_3(\mathsf{d}_5(z)_0)=2$. In this case a private channel is used to communicate a piece of information. The simulator sets the output lock $Lock_o$ and then starts locating the dominion in which the private channel gets interpreted. Before the output lock is released no other output actions at any private channel will be considered by the simulator. This is fine in an interleaving semantics, and would prevent the two consecutive output actions at private channels defined by say $\overline{p}[A].\overline{q}[B].T$ from being simulated in wrong order. The locating procedure interacts with a process of the form $Out(u)$, where u is the number associated to a dominion and it appears as an index for a distinct private channel. The simulator checks if $Out(u)$ is in the right dominion, that is if $\mathsf{d}_5(z)_0=u$. If not S_n continues to interact with another process of the form $Out(\_)$. This procedure should be neither divergent nor deadlocked. The guard l' blocks $Out(u)$ with wrong u once it has been examined. That removes the possibility of $Out(u)$ being examined again and again. The right $Out(\_)$ must be eventually reached. When that happens, the blocked $Out(\_)$'s are freed one by synchronising with the guard $\overline{l'}$. The number stored at m_o indicates how many $Out(\_)$'s are still blocked. The lock $Lock_o$ is released at the same time the last blocked $Out(\_)$ is released, which is indicated by a synchronization at the private channel l. Had $Lock_o$ been released while some $Out(\_)$'s are blocked, it would be possible that a deadlock occurs. After the right dominion is found the simulator can simulate the output action by performing $\overline{o}(\mathsf{d}_5(z)_1)$ of $O(z)$ in that dominion.
3. If $\mathsf{r}_5(z)=4$ then the number codes up a localization process. A dominion is declared to which is attached a number larger than any number associated to any dominion created so far. In this way it is guaranteed that every dominion is associated with a distinguished number.
4. The interpretations in the case of $\mathsf{r}_5(z)=3$ and $\mathsf{r}_5(z)=5$ are simple.

This completes the explanation of the simulator.

5.3. Correctness

An output action says $\overline{a}[A]$ of an $\mathbb{L}$-process is bisimulated by the output action $\overline{a}([\![A]\!]_{\mathfrak{f}})$ of the interpretation. Once the deadlock prevention mechanism defined by $Out(y)$ and $In(y)$ is understood, the interpretation is easy to justify. Formally the correctness of the interpreter and the submodel relationship $\mathbb{L}\sqsubseteq\mathbb{V}$ is proved in one go. By letting

$$[\![P]\!]_{\mathbb{V}}=(p)(\overline{p}[[\![P]\!]_{\mathfrak{f}}]\,|\,\mathcal{I}^{p}),\tag{13}$$

we define a binary relation $[\![\_]\!]_{\mathbb{V}}=\{(P,[\![P]\!]_{\mathbb{V}})\mid P \text{ is a process in } \mathbb{L}\}$. Let $\propto_{\mathbb{V}}$ be the composite relation $[\![\_]\!]_{\mathbb{V}};=_{\mathbb{V}}=\{(P,Q)\mid [\![P]\!]_{\mathbb{V}}=_{\mathbb{V}}Q\}$. We now argue that $\propto_{\mathbb{V}}$ is a subbisimilarity from $\mathbb{L}$ to $\mathbb{V}$.

A context $C[\ldots]$ is defined as follows: (i) A hole $[\_]$ is a context; (ii) $(p)C[\_]$ is a context if $C[\_]$ is; (iii) $C[\_]\,|\,C'[\_]$ is a context if $C[\_]$ and $C'[\_]$ are. If $C[\ldots]$ is a context with k holes, $C[T_1,\ldots,T_k]$ is the term obtained by filling the holes by $T_1,T_2,\ldots,T_k$ in that order. We define inductively a relation $\mathcal{R}$ between $\mathbb{L}$-processes and $\mathbb{V}$-processes in terms of contexts. Starting from the empty set, we include a pair (P,Q) into $\mathcal{R}$ if the following statements are valid:

1. $P=C[R_{k_1},\ldots,R_{k_j}]$ for some context $C[\_]$ in $\mathbb{L}$ and some $\mathbb{L}$-terms $R_{k_1},\ldots,R_{k_j}$.
2. $Q=(\widetilde{l}\widetilde{m}no)\left(S_n\,|\,\overline{m_i}(h)\,|\,\overline{m_o}(h)\,|\,\overline{m}(g)\,|\,L_i\,|\,L_o\,|\,(pq)D[T_{k_1},\ldots,T_{k_j}]\right)$ with $\widetilde{l}=l_il_ol_1l_0$, $\widetilde{m}=m_im_om$, $D[\_]$ being some context in $\mathbb{V}$ and $T_{k_1},\ldots,T_{k_j}$ being some $\mathbb{V}$-terms.

3. $D[\_]$ is obtained from $C[\_]$ as follows: If $C[\_]{=}(q')C'[\_]$ then, for some $k_{i'} \in \{i_1, \dots, i_j\}$, $D[\_]{=}(o)(In(k_{i'}) \,|\, Out(k_{i'}) \,|\, D'[\_])$, and $D'[\_]$ is obtained from $C'[\_]$ in the same way. If $C[\_]{=}C_0[\_] \,|\, C_1[\_]$ then $D[\_]{=}D_0[\_] \,|\, D_1[\_]$ and $D_0[\_], D_1[\_]$ are obtained from $C_0[\_], C_1[\_]$ in the same way.
4. For each $k_{i'} \in \{k_1, \dots, k_j\}$, $T_{k_{i'}} = \overline{n}[[k_{i'}/\sigma(q')][\![S_{k_{i'}}]\!]_{\mathfrak{f}}]$ with appropriate q'.

Clearly $\propto_{\mathbb{V}}$ is equal to the composition $\mathcal{R}; =_{\mathbb{V}}$. It is equipollent because the capacity to perform actions at a particular global channel is preserved by the interpreter. Since all arithmetic calculations are state-preserving operations and the encoding does not introduce any new divergence, it is both codivergent and bisimular. It is extensional because

$$(\_)(\_ \,|\, \overline{m_i}(h) \,|\, \overline{m_o}(h) \,|\, O\,) \,|\, (\_)(\_ \,|\, \overline{m_i}(h') \,|\, \overline{m_o}(h') \,|\, O'\,)$$

is equal to $(\_)(\_ \,|\, \overline{m_i}(h{+}h'{+}1) \,|\, \overline{m_o}(h{+}h'{+}1) \,|\, O'')$ for a suitable term O''. The soundness condition is also met because every number is the index of some abstraction. Hence the soundness of $\mathbb{L}$.

Theorem 5.1. $\mathbb{L} \sqsubseteq \mathbb{V}$.

An interesting corollary of the existence of a universal process for $\mathbb{V}$ is the following.

Corollary 5.2. *$\mathbb{V}$ is equivalent to a variant of $\mathbb{V}$ in which the set of symbols for private channels is finite.*

This is a comfortable property from the viewpoint of implementability. Is it enough to have just two symbols for private channel? The answer is unknown at the moment.

5.4. *$\mathbb{L}$ is a submodel of $\mathbb{R}$*

Much of the programming for the $\mathbb{V}$-interpreter of $\mathbb{L}$ can be carried out in $\mathbb{R}$. What cannot be done in $\mathbb{R}$ is for one process to interpret whatever global channels that appear in an $\mathbb{L}$-process. However if our goal is to show that $\mathbb{L}$ is a submodel of $\mathbb{R}$, it is not necessary to ask for a single $\mathbb{R}$-process to play the role of a universal process. What would be sufficient is a family of processes that collectively play the role of a universal process [11]. More precisely for every finite set $\mathcal{G}$ of global channels, we need a *local interpreter* $\mathcal{I}^{p,\mathcal{G}}$ that is capable of interpreting all $\mathbb{L}$-processes containing only global channels in $\mathcal{G}$. The process $\mathcal{I}^{p,\mathcal{G}}$ is defined by modifying the definition of $\mathcal{I}^{p}$. In $\mathcal{I}^{p,\mathcal{G}}$ the channel function is unnecessary because $\mathcal{I}^{p,\mathcal{G}}$ knows all the global channels in $\mathcal{G}$. Using $\mathcal{I}^{p,\mathcal{G}}$ an encoding of $\mathbb{L}$ in $\mathbb{R}$ can be defined in the fashion of (13), and a subbisimilarity from $\mathbb{L}$ to $\mathbb{R}$ can be constructed.

Theorem 5.3. $\mathbb{L} \sqsubseteq \mathbb{R}$.

The family of local interpreters $\mathcal{I}^{p,\mathcal{G}}$ is uniform in that it is a single process parameterized over $\mathcal{G}$. Consequently the property stated in Corollary 5.2 also holds of $\mathbb{R}$. In fact the finite-private-channel property is valid for any model with a local interpreter.

5.5. *The soundness of the π-calculus*

Looking for the “λ-calculus” of concurrent computation has been an issue in concurrency theory. Early works [40,4,54,55] addressed the issue by introducing higher order mechanism into **CCS** [33]. A crucial shift from the functional paradigm to an object-oriented paradigm, initiated by the work of [8], was fully investigated by Milner, Parrow and Walker with the π-calculus [35]. This is the channel-passing calculus, known as the π-calculus. The π-terms are generated by the following grammar.

$$\begin{aligned} S, T &:= \mathbf{0} \mid \pi.T \mid S \,|\, T \mid (p)T \mid !\mu(x).T \mid !\mu(p).T, \\ \pi &:= \mu(x) \mid \overline{\mu}b \mid \overline{p}\mu \mid \overline{a}p \mid \overline{a}y \mid \overline{x}y \mid \overline{x}p, \end{aligned}$$

where $\mu(p).T$ abbreviates $(p)\mu p.T$. The channel variable x in both $\mu(x).T$ and $!\mu(x).T$ is bound, and the private channel p in both $(p)T$ and $!\mu(p).T$ is closed. We call μ in $\mu(x), \overline{\mu}b$, p in $\overline{p}\mu$, a in $\overline{a}p, \overline{a}y$, and x in $\overline{x}y, \overline{x}p$, *subject names*. A π-process P is a π-term satisfying two conditions: (i) All channel variables are bound and all private channels are closed; (ii) The *unique identifier* condition. To define the second condition, we need to define the concurrent components of a term. A term S is a *concurrent component* of a term T if either $S \equiv T$, or $T \equiv (q)T'$ such that S is a concurrent component of T', or $T \equiv T_1 \,|\, T_2$ such that S is a concurrent component of T_1 or a concurrent component of T_2. We say that a term T satisfies the unique identifier condition if whenever $\overline{a}p.R$ respectively $\overline{x}p.R$ is a concurrent component of T, then the following statements are valid: for every other concurrent component T' of T,

1. either T' does not contain any occurrence of p respectively x,
2. or T' is of the form $p(z).R'$ respectively $x(z).R'$ and in R' all local channels respectively channel variables apart from p respectively x are closed respectively bound.

Our version of the π-calculus differs from the standard one [35] mainly in that the output prefixes are constrained so that there is essentially no potentially insecure scope extrusion like in

$$(p)(\overline{a}p.R \mid \overline{b}p.S) \mid a(x).T \xrightarrow{\tau} (p)(R \mid \overline{b}p.S \mid T\{p/x\}),$$

where an action of $\overline{a}p.R$ reveals the private channel p to T without the consent of $\overline{p}b.S$. Name extrusion is still possible. Bearing in mind that $(p)(\overline{a}p.R \mid \overline{p}c.S \mid S') = \overline{a}(p).(R \mid \overline{p}c.S) \mid S'$ whenever p does not appear in S', the unique identifier condition says that essentially all name extrusions are induced by terms of the form $\overline{a}(p).R$ where the use of the private channel p is restricted to R. As far as we know this form of name extrusion is sufficient in applications.

The action set is $\{\alpha\beta, \overline{p}\beta, \overline{\alpha}b, \overline{\alpha}(p) \mid \alpha, \beta \in \mathcal{C}_g \cup \mathcal{C}_p, b \in \mathcal{C}_g, p \in \mathcal{C}_p\} \cup \{\tau\}$. The labeled transition semantics is defined by the structural, the composition, the replication rules and the following rules, where $\varpi := \mu(x) \mid \mu(p)$.

$$\frac{}{\alpha(x).T \xrightarrow{\alpha\beta} T\{\beta/x\}} \qquad \frac{}{\overline{\alpha}\beta.T \xrightarrow{\overline{\alpha}\beta} T} \qquad \frac{T \xrightarrow{\overline{\alpha}p} T'}{(p)T \xrightarrow{\overline{\alpha}(p)} T'}$$

$$\frac{S \xrightarrow{\overline{\alpha}(p)} S' \quad T \xrightarrow{\alpha p} T'}{S \mid T \xrightarrow{\tau} (p)(S' \mid T')} \qquad \frac{S \xrightarrow{\alpha p} S' \quad T \xrightarrow{\overline{\alpha}(p)} T'}{S \mid T \xrightarrow{\tau} (p)(S' \mid T')} \qquad \frac{\varpi.T \xrightarrow{\lambda} T'}{!\varpi.T \xrightarrow{\lambda} T' \mid !\varpi.T}.$$

The completeness of the π-calculus is proved in [14]. We remark that our unique identifier condition is inconsequential as far as the completeness proof is concerned. The soundness of the π-calculus can be established by recycling the soundness proof for $\mathbb{L}$.

Theorem 5.4. $\mathbb{C} \sqsubset \pi \sqsubset \mathbb{V}$.

The negative result $\mathbb{V} \not\sqsubseteq \pi$ is proved in [14]. See Section 6 for a more general account.

6. Negative results

We now prove some negative results concerning the submodel relationship. The material in this section is from [15].

We first prove a general negative result about the non-interpretability in the π-calculus. Suppose $\mathfrak{D} = \{\lceil 0 \rfloor_a, \lceil 1 \rfloor_a, \lceil 2 \rfloor_a, \ldots, \lceil i \rfloor_a, \ldots\}$ is an infinite set of processes in some model $\mathbb{M}$ such that $i \neq j$ implies

$$\lceil i \rfloor_a \neq \lceil j \rfloor_a. \tag{14}$$

We assume that for every i the process $\lceil i \rfloor_a$ can do one and only one action, which is an output action at channel a, and turns into a process equivalent to $\mathbf{0}$ after the action. In other words,

$$\lceil i \rfloor_a \xrightarrow{\overline{a}(\widehat{i})} = \mathbf{0}, \tag{15}$$

where $\widehat{i}$ is the message released by $\lceil i \rfloor_a$. There is no specific requirement on the messages $\widehat{0}, \widehat{1}, \widehat{2}, \ldots$ apart from that they are pairwise distinct. We require that in $\mathbb{M}$ we can define an absorbing process $\lceil \lambda x.\mathbf{0} \rfloor_a$, a successor process $\lceil \lambda x.x^+ \rfloor_a$, a choice process $\lceil \lambda x.\tau.x^+ + \tau \rfloor_a$, and a test process $\lceil \lambda x.x \overset{?}{>} k \rfloor_a$ for each $k \in \omega$. For each $i \in \omega$ each of the processes can carry out the input action $a(\widehat{i})$, which is the only action the process can perform. Their operational behaviours are specified as follows:

$$\lceil \lambda x.\mathbf{0} \rfloor_a \xrightarrow{a(\widehat{i})} = \mathbf{0}, \tag{16}$$

$$\lceil \lambda x.x^+ \rfloor_a \xrightarrow{a(\widehat{i})} = \lceil i+1 \rfloor_a, \tag{17}$$

$$\lceil \lambda x.\tau.x^+ + \tau \rfloor_a \xrightarrow{a(\widehat{i})} = \tau.\lceil i+1 \rfloor_a + \tau, \tag{18}$$

$$\lceil \lambda x.x \overset{?}{>} k \rfloor_a \xrightarrow{a(\widehat{i})} = \mathbf{0}, \ \text{ if } i \leq k, \tag{19}$$

$$\lceil \lambda x.x \overset{?}{>} k \rfloor_a \xrightarrow{a(\widehat{i})} = \lceil i+1 \rfloor_a, \ \text{ if } i > k. \tag{20}$$

We also require that in $\mathbb{M}$ there is a replicated form for each of $\lceil \lambda x.\mathbf{0} \rfloor_a$, $\lceil \lambda x.x^+ \rfloor_a$ and $\lceil \lambda x.\tau.x^+ + \tau \rfloor_a$, denoted respectively by $\lceil !\lambda x.\mathbf{0} \rfloor_a$, $\lceil !\lambda x.x^+ \rfloor_a$ and $\lceil !\lambda x.\tau.x^+ + \tau \rfloor_a$. For each $i \in \omega$ their unique actions are described as follows:

$$\lceil !\lambda x.\mathbf{0}\rfloor_a \xrightarrow{a(\widehat{i})} = \lceil !\lambda x.\mathbf{0}\rfloor_a, \tag{21}$$

$$\lceil !\lambda x.x^+\rfloor_a \xrightarrow{a(\widehat{i})} = \lceil i{+}1\rfloor_a \,|\, \lceil !\lambda x.x^+\rfloor_a, \tag{22}$$

$$\lceil !\lambda x.\tau.x^+{+}\tau\rfloor_a \xrightarrow{a(\widehat{i})} = (\tau.\lceil i{+}1\rfloor_a + \tau) \,|\, \lceil !\lambda x.\tau.x^+{+}\tau\rfloor_a. \tag{23}$$

Let G be the process $\lceil !\lambda x.\mathbf{0}\rfloor_a \,|\, \lceil !\lambda x.x^+\rfloor_a \,|\, \lceil !\lambda x.\tau.x^+{+}\tau\rfloor_a$. We assume that in $\mathbb{M}$ the following semantic equality is valid for each $k \in \omega$.

$$\Omega \,|\, G = \Omega \,|\, G \,|\, \lceil \lambda x.x \overset{?}{>} k\rfloor_a, \tag{24}$$

where Ω is an $\mathbb{M}$-process whose only action is $\Omega \xrightarrow{\tau} \Omega$. The equality (24) holds because an action of $\lceil \lambda x.x \overset{?}{>} k\rfloor_a$ can be bisimulated by either $\lceil !\lambda x.\mathbf{0}\rfloor_a$ or $\lceil !\lambda x.x^+\rfloor_a$. Our assumption on $\mathbb{M}$ is very liberal. It asks for no more than a divergent process Ω and a numerical system that validates (24). A concrete example is given in the proof of Theorem 6.2. Before continuing, the reader could take a look at the concrete example for intuition.

Theorem 6.1. $\mathbb{M} \not\sqsubseteq \pi$.

Proof. Suppose there were a subbisimilarity $\Im$ from the $\mathbb{M}$-processes to the π-processes. In what follows we write $M \Im P \not\rightarrow$ for $M \Im P$ and $P \not\rightarrow$. Now fix $a \in \mathcal{C}_g$. We fix the notations for the interpretations of the $\mathbb{M}$-processes just described. We remark that since $\lceil i\rfloor_a$ for example cannot do any internal action, by codivergence we may assume that N_i cannot do any internal action.

- For each $i \in \omega$ let N_i be the π-process such that $\lceil i\rfloor_a \Im N_i \not\rightarrow$.
- Let C_0 be the π-process such that $\lceil \lambda x.\mathbf{0}\rfloor_a \Im C_0 \not\rightarrow$.
- Let S_0 be the π-process such that $\lceil \lambda x.x^+\rfloor_a \Im S_0 \not\rightarrow$.
- Let E_0 be the π-process such that $\lceil \lambda x.\tau.x^+{+}\tau\rfloor_a \Im E_0 \not\rightarrow$.
- Let C be the π-process such that $\lceil !\lambda x.\mathbf{0}\rfloor_a \Im C \not\rightarrow$.
- Let S be the π-process such that $\lceil !\lambda x.x^+\rfloor_a \Im S \not\rightarrow$.
- Let E be the π-process such that $\lceil !\lambda x.\tau.x^+{+}\tau\rfloor_a \Im E \not\rightarrow$.
- For each $k \in \omega$ let J_k be the π-process such that $\lceil \lambda x.x \overset{?}{>} k\rfloor_a \Im J_k \not\rightarrow$.

It follows from (24) and extensionality that the following equality holds for every $k \in \omega$.

$$\Omega \,|\, C \,|\, S \,|\, E = \Omega \,|\, C \,|\, S \,|\, E \,|\, J_k. \tag{25}$$

We now derive some properties for the π-processes $N_0, N_1, N_2, N_3, \ldots$.

1. We argue that $S_0 \,|\, N_i \xrightarrow{\iota} = N_{i+1}$ is essentially the only one-step nondeterministic computation of $S_0 \,|\, N_i$. The following argument depends crucially on the fact that neither S_0 nor N_i can do any internal action. According to codivergence property there is no infinite computation sequence from $S_0 \,|\, N_i$. Suppose $S_0 \,|\, N_i \rightarrow B$. This action must be caused by $S_0 \xrightarrow{\lambda_a} S'$ and $N_i \xrightarrow{\overline{\lambda_a}} N'$ for some complementary actions $\lambda_a, \overline{\lambda_a}$ and some π-processes S' and N'. If neither λ_a nor $\overline{\lambda_a}$ is a bound output action then $S_0 \,|\, N_i \xrightarrow{\lambda_a}\xrightarrow{\overline{\lambda_a}} B$ and $S_0 \,|\, N_i \xrightarrow{\overline{\lambda_a}}\xrightarrow{\lambda_a} B$. It follows that $B \rightarrow^* \xrightarrow{\lambda_a} \rightarrow^* \xrightarrow{\overline{\lambda_a}} B' = B$ and $B \rightarrow^* \xrightarrow{\overline{\lambda_a}} \rightarrow^* \xrightarrow{\lambda_a} B'' = B$ for some B', B''. Now $B \,|\, B \rightarrow^+ B' \,|\, B'' = B \,|\, B$. We derive by induction that there would be an infinite computation sequence from $S_0 \,|\, N_i \,|\, S_0 \,|\, N_i$, contradicting to the fact that $\lceil \lambda x.x^+\rfloor_a \,|\, \lceil i\rfloor_a \,|\, \lceil \lambda x.x^+\rfloor_a \,|\, \lceil i\rfloor_a$ is not divergent. If one of $\lambda_a, \overline{\lambda_a}$ is a bound output action, we also get an infinite computation by renaming and inserting scope operators in appropriate places. More specifically without loss of generality suppose $\lambda_a = ao$ and $\overline{\lambda_a} = \overline{a}(o)$. Let $S_0 \xrightarrow{aq} B_1$ and $N_i \xrightarrow{\overline{a}(p)} B_2$ for fresh private channels p, q. Then

$$B \equiv (o)(B_1\{o/p\} \,|\, B_2\{o/q\}). \tag{26}$$

It follows from $S_0 \,|\, N_i = B$ that there must exist some B' such that

$$S_0 \,|\, N_i \xrightarrow{aq}\xrightarrow{\overline{a}(p)} B_1 \,|\, B_2 \tag{27}$$

is bisimulated by

$$B \rightarrow^* \xrightarrow{aq} \rightarrow^* \xrightarrow{\overline{a}(p)} B'. \tag{28}$$

Now (26) and (28) imply that (27) can be extended to

$$S_0 \,|\, N_i \xrightarrow{aq}\xrightarrow{\overline{a}(p)} B_1 \,|\, B_2 \Longrightarrow\xrightarrow{\lambda_a^1}\Longrightarrow\xrightarrow{\overline{\lambda_a^1}} B_1' \,|\, B_2' \tag{29}$$

for some B_1', B_2' and some $\lambda_a^1, \overline{\lambda_a^1}$. Consequently the bisimulation (28) can be extended to

$$B \rightarrow^* \xrightarrow{aq} \rightarrow^* \xrightarrow{\overline{a}(p)} B' \Longrightarrow\xrightarrow{\lambda_a^1}\Longrightarrow\xrightarrow{\overline{\lambda_a^1}} B'' \tag{30}$$

for some B''. The extension can be repeated infinitely often. We eventually get an infinite sequence with alternating input-output actions. Similarly we can derive from $S_0 \,|\, N_i \xrightarrow{\overline{a}(p)}\xrightarrow{aq} B_1 \,|\, B_2$ an infinite sequence with alternating output-input actions. In this way $S_0 \,|\, N_i \,|\, S_0 \,|\, N_i$ would induce an infinite sequence of computation. This is again a contradiction. We are done.

2. We are going to prove that one and only one of the following may happen: (i) All N_i's can only do input actions at a; (ii) Almost all N_i's can only do bound output actions at a. The proof is as follows:
   (a) N_i cannot do both an input action at channel a and an output action at channel a. Otherwise $N_i \,|\, N_i$ would be able to do an interaction, which would be a contradiction. This is because $N_i \,|\, N_i$ cannot perform any nondeterministic computation since $\lceil i \rfloor_a \,|\, \lceil i \rfloor_a$ cannot do any internal action. It cannot do a deterministic computation step since that would induce an infinite sequence of internal actions from $N_i \,|\, N_i \,|\, N_i \,|\, N_i$, like in the previous case. So N_i may perform either an input action or an output action exclusively.
   (b) If N_i can do an input, respectively output action then N_j can do an input, respectively output action for all $j \in \omega$ since the latter has to interact with C_0.
   (c) It follows from $\lceil \lambda x.\mathbf{0} \rfloor_a \,|\, \lceil i \rfloor_a \xrightarrow{\iota} = \mathbf{0}$ that $C_0 \,|\, N_i \xrightarrow{\iota} = \mathbf{0}$. Suppose $N_i \xrightarrow{\overline{a}c} N'$ and $N_j \xrightarrow{\overline{a}c} N''$ for some N', N''. Clearly $N' = \mathbf{0} = N''$. But then one could derive from $N_i \,|\, S_0 \xrightarrow{\iota} = N_{i+1}$ and $N_j \,|\, S_0 \xrightarrow{\iota} = N_{j+1}$ that $N_{i+1} = N_{j+1}$, which implies $i = j$. We conclude that if both N_i and N_j can do free output actions at channel a then the channels they release must be distinct whenever $i \neq j$.
   (d) According to our assumption we have $S_0 \,|\, N_i \xrightarrow{\tau} = N_{i+1}$ for all $i \geq 0$. Let's write $P \xrightarrow{\lambda}$ if $P \xrightarrow{\lambda} P'$ for some P'. It should be clear that if $N_1 \xrightarrow{\overline{a}c}$, then $S_0 \,|\, N_0 \Longrightarrow\xrightarrow{\overline{a}c}$. Similarly if $N_2 \xrightarrow{\overline{a}d}$, then $S_0 \,|\, N_1 \Longrightarrow\xrightarrow{\overline{a}d}$. Thus $S_0 \,|\, S_0 \,|\, N_0 \Longrightarrow\xrightarrow{\overline{a}d}$. By induction we can prove that if N_i can release a global channel at a then that global channel must appear in $S_0 \,|\, N_0$. So only a finite number of $N_0, N_1, N_2, \ldots$ can do free output actions. Let h be the least number such that, for every $j \geq h$, N_j does only a bound output action.

By a case analysis on the actions of $N_0, N_1, N_2, N_3, \ldots$ we can prove the impossibility result.

1. Suppose $k > h$. Let

$$N_h \xrightarrow{\overline{a}(p)} N_h',$$
$$N_k \xrightarrow{\overline{a}(p)} N_k'.$$

In this case C_0, D_0, E_0, J_h, C, D and E can only do input actions at channel a. Let

$$C_0 \xrightarrow{ap} C_0^p,$$
$$D_0 \xrightarrow{ap} S_0^p,$$
$$E_0 \xrightarrow{ap} E_0^p,$$
$$J_h \xrightarrow{ap} J_p,$$
$$C \xrightarrow{ap} C_p = C,$$
$$D \xrightarrow{ap} S_p = S_0^p \,|\, S,$$
$$E \xrightarrow{ap} E_p = E_0^p \,|\, E.$$

None of C_0, D_0, E_0, J_h, C, D and E may perform any output actions. Otherwise there would be either an infinite deterministic computation or a nondeterministic computation step. It follows from (25) that $\Omega \,|\, C \,|\, S \,|\, E \,|\, J_h \xrightarrow{ap} \Omega \,|\, C \,|\, S \,|\, E \,|\, J_p$ must be bisimulated by one of the following.

$$\Omega \,|\, C \,|\, S \,|\, E \xrightarrow{ap} \Omega \,|\, C_p \,|\, S \,|\, E,$$
$$\Omega \,|\, C \,|\, S \,|\, E \xrightarrow{ap} \Omega \,|\, C \,|\, S_p \,|\, E,$$
$$\Omega \,|\, C \,|\, S \,|\, E \xrightarrow{ap} \Omega \,|\, C \,|\, S \,|\, E_p.$$

In the first case $\Omega\,|\,C\,|\,S\,|\,E\,|\,J_h\,|\,N_k \xrightarrow{\iota} (p)(\Omega\,|\,C\,|\,S\,|\,E\,|\,J_p\,|\,N'_k)$ should be bisimulated by $\Omega\,|\,C\,|\,S\,|\,E\,|\,N_k \xrightarrow{\iota} (p)(\Omega\,|\,C_p\,|\,S\,|\,E\,|\,N'_k)$ due to congruence. This is impossible because, according to Lemma 2.1,

$$\begin{aligned}(p)(\Omega\,|\,C\,|\,S\,|\,E\,|\,J_p\,|\,N'_k) &= \Omega\,|\,C\,|\,S\,|\,E\,|\,N_{k+1}\\ &\neq \Omega\,|\,C\,|\,S\,|\,E\\ &= (p)(\Omega\,|\,C\,|\,S\,|\,E\,|\,N'_k)\\ &= (p)(\Omega\,|\,C_p\,|\,S\,|\,E\,|\,N'_k).\end{aligned}$$

In the second case $\Omega\,|\,C\,|\,S\,|\,E\,|\,J_h\,|\,N_h \xrightarrow{\iota} (p)(\Omega\,|\,C\,|\,S\,|\,E\,|\,J_p\,|\,N'_h)$ should be bisimulated by $\Omega\,|\,C\,|\,S\,|\,E\,|\,N_h \xrightarrow{\iota} (p)(\Omega\,|\,C\,|\,S_p\,|\,E\,|\,N'_h)$. Using again the full abstraction lemma one derives the following contradiction.

$$\begin{aligned}(p)(\Omega\,|\,C\,|\,S\,|\,E\,|\,J_p\,|\,N'_h) &= \Omega\,|\,C\,|\,S\,|\,E\\ &\neq \Omega\,|\,C\,|\,S\,|\,E\,|\,N_{h+1}\\ &= (p)(\Omega\,|\,C\,|\,S^p_0\,|\,S\,|\,E\,|\,N'_h)\\ &= (p)(\Omega\,|\,C\,|\,S_p\,|\,E\,|\,N'_h).\end{aligned}$$

In the third case $\Omega\,|\,C\,|\,S\,|\,E\,|\,J_h\,|\,N_k \xrightarrow{\iota} (p)(\Omega\,|\,C\,|\,S\,|\,E\,|\,J_p\,|\,N'_k)$ need be bisimulated by $\Omega\,|\,C\,|\,S\,|\,E\,|\,N_k \xrightarrow{\iota} (p)(\Omega\,|\,C\,|\,S\,|\,E_p\,|\,N'_k)$. This is also impossible because $(p)(\Omega\,|\,C\,|\,S\,|\,E_p\,|\,N'_k) = (p)(\Omega\,|\,C\,|\,S\,|\,E^p_0\,|\,E\,|\,N'_k)$ can do a nondeterministic computation step, reaching to a state where N_{k+1} is *not* a concurrent component, whereas on the other hand the equality $(p)(\Omega\,|\,C\,|\,S\,|\,E\,|\,J_p\,|\,N'_k) = \Omega\,|\,C\,|\,S\,|\,E\,|\,N_{k+1}$ is valid.

2. Now suppose the immediate actions of $N_0, N_1, N_2, \ldots$ are input actions. In this case an output action of $\Omega\,|\,C\,|\,S\,|\,E\,|\,J_h$ induced by J_h, whether it is a free output action or a bound output action, must be bisimulated by an output action of $\Omega\,|\,C\,|\,S\,|\,E$ induced by C or D or E. We can deduce a contradiction too.

We have proved that there cannot be any subbisimilarity from $\mathbb{M}$ to the π-calculus. $\square$

We can now derive some negative results concerning the π-calculus. The intuition for the negative results is that in each of $\mathbb{T}$, $\mathbb{L}$, $\mathbb{R}$ and $\mathbb{V}$ one can define processes $\lceil !\lambda x.\mathbf{0}\rfloor_a$, $\lceil !\lambda x.x^+\rfloor_a$, $\lceil !\lambda x.\tau.x^+ + \tau\rfloor_a$ and $\lceil \lambda x.x \overset{?}{>} k\rfloor_a$ rendering the equality (24) valid. One can also define these processes in π, but the equality (24) is *not* valid. The reason is that in π when the process $\lceil \lambda x.x \overset{?}{>} k\rfloor_a$ receives the encoding of a number, it inputs a name that points to the 'number'. At this point it does not know what the number is. For the same reason it is impossible for $\Omega\,|\,\lceil !\lambda x.\mathbf{0}\rfloor_a\,|\,\lceil !\lambda x.x^+\rfloor_a\,|\,\lceil !\lambda x.\tau.x^+ + \tau\rfloor_a$ to know how to simulate the action.

Theorem 6.2. *$\mathbb{T} \not\sqsubseteq \pi$, $\mathbb{L} \not\sqsubseteq \pi$, $\mathbb{R} \not\sqsubseteq \pi$ and $\mathbb{V} \not\sqsubseteq \pi$.*

Proof. In view of Theorem 4.2 it suffices to prove that $\mathbb{T} \not\sqsubseteq \pi$ and $\mathbb{L} \not\sqsubseteq \pi$. We argue that $\mathbb{L} \not\sqsubseteq \pi$. The proof of $\mathbb{T} \not\sqsubseteq \pi$ is similar. To start with we define the encoding of the numbers in $\mathbb{L}$. Given an abstraction A we write A^+ for $(x,y)\overline{x}[A]$. Let

$$\begin{aligned}\widehat{0} &\overset{\text{def}}{=} (x,y)\overline{y}[\mathbf{0}],\\ \widehat{k+1} &\overset{\text{def}}{=} (x,y)\overline{x}[\widehat{k}^+].\end{aligned}$$

The abstractions $\widehat{0}, \widehat{1}, \widehat{2}, \ldots$ represent numbers. Since $\mathbb{L}$ is complete, computable manipulations of these numbers can be defined in $\mathbb{L}$. To apply the general negative result it suffices to define the following simple processes.

$$\begin{aligned}\lceil i\rfloor_a &\overset{\text{def}}{=} \overline{a}[\widehat{i}],\\ \lceil \lambda x.\mathbf{0}\rfloor_a &\overset{\text{def}}{=} a(X),\\ \lceil \lambda x.x^+\rfloor_a &\overset{\text{def}}{=} a(X).\overline{a}[X^+],\\ \lceil \lambda x.\tau.x^+ + \tau\rfloor_a &\overset{\text{def}}{=} a(X).\left(\tau.\overline{a}[X^+] + \tau\right),\\ \lceil \lambda x.x \overset{?}{>} k\rfloor_a &\overset{\text{def}}{=} a(X).(n_1o_1)(X(n_1,o_1)\,|\,n_1(X_1).(n_2o_2)(X_1(n_2,o_2)\,|\,\ldots\\ &\qquad |\,n_{k-1}(X_{k-1}).(n_ko_k)(X_{k-1}(n_k,o_k)\,|\,n_k.\overline{a}[X^+])\ldots)),\\ \lceil !\lambda x.\mathbf{0}\rfloor_a &\overset{\text{def}}{=} (p)(a(X).p(Z).(Z(a,p)\,|\,\overline{p}[Z])\\ &\qquad |\,\overline{p}[(x,z)x(X).z(Z).(Z(x,z)\,|\,\overline{z}[Z])]),\end{aligned}$$

$$\begin{aligned}\lceil !\lambda x.x^+\rfloor_a &\overset{\text{def}}{=} (p)(a(X).(\overline{a}[X^+]\,|\,p(Z).(Z(a,p)\,|\,\overline{p}[Z]))\\ &\qquad |\,\overline{p}[(x,z)x(X).(\overline{x}[X^+]\,|\,z(Z).(Z(x,z)\,|\,\overline{z}[Z])])),\\ \lceil !\lambda x.\tau.x^+{+}\tau\rfloor_a &\overset{\text{def}}{=} (p)(a(X).((\tau.\overline{a}[X^+]+\tau)\,|\,p(Z).(Z(a,p)\,|\,\overline{p}[Z]))\\ &\qquad |\,\overline{p}[(x,z)x(X).((\tau.\overline{x}[X^+]+\tau)\,|\,z(Z).(Z(x,z)\,|\,\overline{z}[Z])])).\end{aligned}$$

The last three processes, $\lceil !\lambda x.\mathbf{0}\rfloor_a$, $\lceil !\lambda x.x^+\rfloor_a$ and $\lceil !\lambda x.\tau.x^+{+}\tau\rfloor_a$, are intuitively the processes $!a(X)$, $!a(X).\overline{a}[X^+]$ and $!a(X).\big(\tau.\overline{a}[X^+]+\tau\big)$. These replication processes must be implemented in $\mathbb{L}$. It is easy to verify that the processes defined in the above satisfy the necessary requirements stated in (14) through (23).

Let $G \overset{\text{def}}{=} \lceil !\lambda x.\mathbf{0}\rfloor_a\,|\,\lceil !\lambda x.x^+\rfloor_a\,|\,\lceil !\lambda x.\tau.x^+{+}\tau\rfloor_a$. We prove that $\Omega\,|\,G = \Omega\,|\,G\,|\,\lceil \lambda x.x \overset{?}{>} k\rfloor_a$. Consider the following transition

$$\Omega\,|\,G\,|\,\lceil \lambda x.x \overset{?}{>} k\rfloor_a \xrightarrow{a(A)} \Omega\,|\,G\,|\,J_A, \tag{31}$$

where J_A is the following process

$$(n_1o_1)(A(n_1,o_1)\,|\,n_1(X_1).(n_2o_2)(X_1(n_2,o_2)\,|\,\ldots\,|\,(X_{k-1}(n_k,o_k)\,|\,n_k.\overline{a}[A^+])\ldots)).$$

The process J_A and all its descendants can either carry out an internal action or enable $\overline{a}[A^+]$. They cannot do any other actions. Let's call P' a τ descendant of P if $P \Longrightarrow P'$. Let $\mathfrak{J}$ be the set of all τ descendants of J_A. It can be partitioned into three disjoint subsets.

$$\begin{aligned}\mathfrak{J}_0 &\overset{\text{def}}{=} \{J' \in \mathfrak{J} \mid \text{no } \tau \text{ descendant of } J' \text{ can enable } \overline{a}[A^+]\},\\ \mathfrak{J}_1 &\overset{\text{def}}{=} \{J' \in \mathfrak{J} \mid \text{every } \tau \text{ descendant of } J' \text{ can enable } \overline{a}[A^+] \text{ potentially}\},\\ \mathfrak{J}_{\frac{1}{2}} &\overset{\text{def}}{=} \mathfrak{J} \setminus (\mathfrak{J}_0 \cup \mathfrak{J}_1).\end{aligned}$$

If $J_A \in \mathfrak{J}_0$ then (31) can be bisimulated by $\Omega\,|\,G \xrightarrow{a(A)}{=} \Omega\,|\,G$ by invoking the component $\lceil !\lambda x.\mathbf{0}\rfloor_a$. Notice that J_A may induce an infinite computation. But this is not a problem in the presence of Ω. If $J_A \in \mathfrak{J}_1$ then (31) can be bisimulated by $\Omega\,|\,G \xrightarrow{a(A)}{=} \Omega\,|\,G\,|\,\overline{a}[A^+]$ by invoking the component $\lceil !\lambda x.x^+\rfloor_a$. If $J_A \in \mathfrak{J}_{\frac{1}{2}}$ then (31) can be bisimulated by $\Omega\,|\,G \xrightarrow{a(A)}{=} \Omega\,|\,G\,|\,(\tau.\overline{a}[A^+]+\tau)$ by invoking the component $\lceil \lambda x.\tau.x^+{+}\tau\rfloor_a$. We only have to prove that every $J' \in \mathfrak{J}_{\frac{1}{2}}$ renders true the following equality.

$$\Omega\,|\,(\tau.\overline{a}[A^+]+\tau) = \Omega\,|\,J'. \tag{32}$$

Suppose $J' \xrightarrow{\tau} J''$. It induces the transition

$$\Omega\,|\,J' \xrightarrow{\tau} \Omega\,|\,J''. \tag{33}$$

If $J'' \in \mathfrak{J}_{\frac{1}{2}}$, then (33) is bisimulated by $\Omega\,|\,(\tau.\overline{a}[A^+]+\tau) \xrightarrow{\tau} \Omega\,|\,(\tau.\overline{a}[A^+]+\tau)$. If $J'' \in \mathfrak{J}_0$, then (33) is bisimulated by $\Omega\,|\,(\tau.\overline{a}[A^+]+\tau) \xrightarrow{\tau} \Omega\,|\,\mathbf{0}$. If $J'' \in \mathfrak{J}_1$, then (33) is bisimulated by $\Omega\,|\,(\tau.\overline{a}[A^+]+\tau) \xrightarrow{\tau} \Omega\,|\,\overline{a}[A^+]$. We are done. □

The result $\mathbb{L} \not\sqsubseteq \pi$ seems to contradict to the well-known encoding of higher order π-calculus π^ω in the first order π-calculus [49,50] and other similar encodings [54,58]. The higher order process calculi studied in these encodings are both abstraction-passing and complete. The criteria for relative expressiveness adopted in these papers are weaker than the one of this paper. They include extensionality, equipollence and *weak* bisimulation. Some of the encodings satisfy the full abstraction property, others satisfy only a weaker form of it. Almost all encodings [54,58] satisfy the codivergence and branching bisimulation property. As criteria for submodel relationship codivergence and branching bisimulation help to derive reasonable properties about interpretations, which is duly demonstrated in the proof of Theorem 6.1. Without these two criteria we are not able to conclude for example that N_i cannot perform both an input action and an output action. The source models considered in the above mentioned papers are more liberal than $\mathbb{L}$. The equality stated in (24) fails in π^ω since an observer can be powerful enough to detect the presence of the component $\lceil \lambda x.x \overset{?}{>} k\rfloor_a$. Thus Theorem 6.2 does not contradict to the expressiveness results given by the encodings in the afore mentioned papers. We would however like to draw reader's attention to the fact that if we impose the same restrictions on the abstractions in π^ω we get a variant, denoted by say $\pi^\omega_\emptyset$, that is comparable to $\mathbb{L}$. The proof of Theorem 6.2 applies to $\pi^\omega_\emptyset$. It follows that $\pi^\omega_\emptyset \not\sqsubseteq \pi$. On the other hand Sangiorgi's encoding of π^ω in π is also an encoding of $\pi^\omega_\emptyset$ in π. What we do not understand at the moment is if the encoding is still fully abstract. Notice that $\pi^\omega_\emptyset \sqsubseteq \pi^\omega$ would imply $\pi^\omega \not\sqsubseteq \pi$. We do not expect that the question "$\pi^\omega_\emptyset \sqsubseteq \pi^\omega$?" is easy to answer. A possible way to attack problems of this kind is to explore the power of universal processes [11]. This is left for future investigation.

In the same fashion the result $\mathbb{T} \not\sqsubseteq \pi$ seems to contradict to the result in [32] stating that the behaviour of reactive Turing machines can be simulated in the π-calculus up to divergence-preserving branching bisimilarity. The main reason

$$\begin{tikzcd}
& & \mathbb{V} & & \\
& \mathbb{R} \arrow[ur] & & & \\
\mathbb{L} \arrow[ur] & & \mathbb{T} \arrow[ul] & & \pi \arrow[uull] \\
& & \mathbb{C} \arrow[ull] \arrow[u] \arrow[urr] & &
\end{tikzcd}$$

Fig. 2. Submodel Relatiohship.

is that the reactive Turing machines have a weaker communication mechanism, and consequently a stronger distinguishing power. Messages are passed around token by token, which is easy for the π-calculus to simulate.

Fig. 2 summarises our current understanding of the submodel relationship among the six models. Notice that $\mathbb{L} \not\sqsubseteq \mathbb{T}$ since $!a(X).\overline{b}[X]$ cannot be interpreted in $\mathbb{T}$. Notice also that $\pi \not\sqsubseteq \mathbb{R}$ because $a(x).\overline{x}x$ is not interpretable by $\mathbb{R}$. It follows that $\pi \not\sqsubseteq \mathbb{L}$ and $\pi \not\sqsubseteq \mathbb{T}$. The issue that has not been resolved is if $\mathbb{T} \sqsubseteq \mathbb{L}$. We remark that $\mathbb{T} \not\sqsubseteq \mathbb{L}$ implies $\mathbb{R} \not\sqsubseteq \mathbb{L}$. The diagram in Fig. 2 is drawn by assuming $\mathbb{T} \not\sqsubseteq \mathbb{L}$.

7. Conclusion

Axiom of Completeness ($\forall \mathbb{M}.\, \mathbb{C} \sqsubseteq \mathbb{M}$) was proposed as a foundational postulate in [14]. However a model satisfying Axiom of Completeness could be too powerful to be implementable. This open-ended view is replaced by a closed world view in the present paper. What we have done in this paper is to provide evidence for the universal validity of the Thesis of Interaction. The three classical computation models can all be lifted in a natural way to interaction models that satisfy the Thesis of Interaction.

Being a powerful model, $\mathbb{V}$ has only a few primitives. In $\mathbb{V}$ nondeterminism is solely due to interaction. No intensional nondeterminism is admitted in any form. It is easy to convince oneself that a guarded choice, say $\overline{a}(0).P + \overline{b}(1).Q$, cannot be defined in $\mathbb{V}$ without introducing divergence. This is established for several variants of the π-calculus [39,42,16], and the proof can be repeated for $\mathbb{V}$. What it indicates is that the guarded choice operator is not implementable. The match and the mismatch operators in the π-calculus are implementable. Our $\mathbb{V}$-interpreter of the π-calculus of this paper can be easily extended to account for them.

From a programming language viewpoint a subbisimilarity $\mathbb{M} \sqsubseteq \mathbb{N}$ is often given by an encoding. Formally an encoding from $\mathbb{M}$ to $\mathbb{N}$ is an effective function $\mathfrak{e}$ from the $\mathbb{M}$-processes to the $\mathbb{N}$-processes such that $\mathfrak{e}; =_{\mathbb{N}}$ is a subbisimilarity. Let's write $\mathbb{M} \sqsubseteq_e \mathbb{N}$ if there is an encoding from $\mathbb{M}$ to $\mathbb{N}$. A programming practitioner would believe the following.

Effective Thesis of Interaction. $\forall \mathbb{M}.\, \mathbb{C} \sqsubseteq_e \mathbb{M} \sqsubseteq_e \mathbb{V}$.

This is the extension of the Effective Church-Turing Thesis. Now suppose there is an encoding $\mathfrak{e}$ from a complete model $\mathbb{M}$ to $\mathbb{V}$. Then there is a $\mathbb{V}$-process that translates the index of an $\mathbb{M}$-process to the index of a $\mathbb{V}$-process. Composing this translation with the universal process for $\mathbb{V}$ one gets a $\mathbb{V}$-interpreter of $\mathbb{M}$. So in the programming world a complete model is sound if and only if it has a $\mathbb{V}$-interpreter.

The Thesis of Interaction is only the first step of the theory of interaction. To explain possible future research directions, we need to talk about processes that do not depend on any particular global channels. A parametric definition is defined by an equation

$$D(x_1,\ldots,x_k) = T, \tag{34}$$

where $x_1,\ldots,x_k$ are variables for natural numbers. T contains no more free variables than $x_1,\ldots,x_k$ and may contain instantiated occurrences of D of the form $D(t_1,\ldots,t_k)$. We say that $D(x_1,\ldots,x_k)$ is a *k-ary parametric definition* defined by (34). The parameters are values. The semantics of $D(x_1,\ldots,x_k)$ are defined by the following rule.

$$\frac{T\{t_1/x_1,\ldots,t_k/x_k\} \xrightarrow{\lambda} T'}{D(t_1,\ldots,t_k) \xrightarrow{\lambda} T'}$$

We call $D(t_1,\ldots,t_k)$ the instantiation of D at $t_1,\ldots,t_k$. In (34) D may appear in T. The unfolding of a parametric definition admits dynamic binding in the sense that a private channel may get captured by a scope operator. Two k-ary parametric definitions $D(x_1,\ldots,x_k)$ and $D'(x_1,\ldots,x_k)$ are equal if $D(i_1,\ldots,i_k) =_{\mathbb{V}} D'(i_1,\ldots,i_k)$ for all numbers $i_1,\ldots,i_k$.

Parametric definition can be defined in terms of replication [11]. A posteriori this has to be true if $\mathbb{V}$ is the maximal model. Now we may as well let parametric definitions be the first class citizens in $\mathbb{V}$. In a straightforward manner we define for each $k > 0$ a Gödel encoding for the k-ary parametric definitions. Using standard technique it is routine to prove the $\mathbb{V}$-version of the fundamental theorems of recursion theory [48,53].

Enumeration Theorem. There is a $k+1$-ary parametric definition $\mathcal{D}_k(x, x_1, \ldots, x_k)$ such that $\mathcal{D}_k(x, x_1, \ldots, x_k)$ is equal to the i-th k-ary parametric definition for all i.

S-m-n Theorem. There is an injective primitive recursive $k+1$-ary function s such that $\mathcal{D}_k(x_1, \ldots, x_m, \ldots, x_{m+n})$ is equal to $\mathcal{D}_{s(k,x_1,\ldots,x_m)}(x_{m+1}, \ldots, x_{m+n})$.

Recursion Theorem. Let f be a total computable function. There is a number n such that $\mathcal{D}_n(x) = \mathcal{D}_{f(n)}(x)$.

Using these theorems a definability theory can be developed for every interaction model.

The Thesis of Interaction also provides the foundation for the theory of nondefinability of interaction models. No process can exhibit the behavior specified in (1) in any interaction model because such a process can not be defined in $\mathbb{V}$. Nondefinability or unsolvability in the setting of concurrency theory is an avenue of research that calls for further efforts.

Declaration of competing interest

There is no conflict of interest.

Acknowledgement

We acknowledge NSFC (62072299, 61772336) for financial support. We thank Prof. Sangiorgi for his instructive comment on the results and the proofs in Section 6. We also thank the members of BASICS for discussions. Finally but not the least, we thank the two anonymous referees for detailed comments and instructive suggestions. Their efforts have greatly improved the quality of this paper.

References

[1] S. Abramsky, The Lazy lambda calculus, in: D. Turner (Ed.), Declarative Programming, Addison-Wesley, 1988, pp. 65–116.
[2] H. Barendregt, The Lambda Calculus: Its Syntax and Semantics, North-Holland, 1984.
[3] J. Baeten, B. Luttik, P. van Tilburg, Reactive Turing machines, Inf. Comput. 231 (2013) 143–166.
[4] G. Boudol, in: Towards a Lambda Calculus for Concurrent and Communicating Systems, TAPSOFT'89, in: Lecture Notes in Comuter Science, vol. 351, 1989, pp. 149–161.
[5] L. Cardelli, A. Gordon, Mobile ambients, Theor. Comput. Sci. 240 (2000) 177–213.
[6] A. Church, An unsolvable problem of elementary number theory, Am. J. Math. 58 (1936) 345–363.
[7] D. Deutsch, Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer, Proceedings of the Royal Society of London, 1985.
[8] U. Engberg, M. Nielsen, A Calculus of Communicating Systems with Label Passing, Report DAIMI PB-208, Computer Science Department, University of Aarhus, 1986.
[9] Wan J. Fokkink, Rob van Glabbeek, Bas Luttik, Divide and congruence III: from decomposition of modal formulas to preservation of stability and divergence, Inf. Comput. 268 (2019).
[10] Y. Fu, Z. Yang, Tau laws for Pi calculus, Theor. Comput. Sci. 308 (2003) 55–130.
[11] Y. Fu, The universal process, Log. Methods Comput. Sci. 13 (2017) 1–23.
[12] Y. Fu, The value-passing calculus, in: Theories of Programming and Formal Methods, in: Lecture Notes in Computer Science, vol. 8051, 2013, pp. 166–195.
[13] Y. Fu, Nondeterministic structure of computation, Math. Struct. Comput. Sci. 25 (2015) 1295–1338.
[14] Y. Fu, Theory of interaction, Theor. Comput. Sci. 611 (2016) 1–49.
[15] Y. Fu, On the power of name-passing communication, in: 28th International Conference on Concurrency Theory (CONCUR 2017), 2017, pp. 22:1–22:15.
[16] Y. Fu, H. Lu, On the expressiveness of interaction, Theor. Comput. Sci. 411 (2010) 1387–1451.
[17] J. Gill, Computational complexity of probabilistic Turing machines, SIAM J. Comput. 6 (1977) 675–695.
[18] R. van Glabbeek, in: C. Baier, U. Dal Lago (Eds.), Proceedings 21st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2018), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2018), Thessaloniki, Greece, April 2018, in: LNCS, vol. 10803, Springer, 2018, pp. 183–202.
[19] R. van Glabbeek, B. Luttik, N. Trcka, Branching bisimilarity with explicit divergence, Fundam. Inform. 93 (4) (2009) 371–392.
[20] Rob J. van Glabbeek, Bas Luttik, Nikola Trcka, Computation tree logic with deadlock detection, Log. Methods Comput. Sci. 5 (4) (2009).
[21] R. van Glabbeek, W. Weijland, Branching time and abstraction in bisimulation semantics, in: Information Processing'89, North-Holland, 1989, pp. 613–618.
[22] R. van Glabbeek, W. Weijland, Branching time and abstraction in bisimulation semantics, J. ACM 43 (3) (1996) 555–600.
[23] K. Gödel, Über Formal Unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme, Monatshefte Mat. Verwand. Syst. I 38 (1931) 173–198.
[24] D. Goldin, S. Smolka, P. Attie, E. Sonderegger, Turing machines, transition systems, and interaction, Inf. Comput. 194 (2) (2004) 101–128.
[25] D. Gorla, Comparing communication primitives via their relative expressive power, Inf. Comput. 206 (2008) 931–952.
[26] D. Gorla, Towards a unified approach to encodability and separation results for process calculi, in: CONCUR 2008, in: Lecture Notes in Computer Science, vol. 5201, 2008, pp. 492–507.
[27] M. Hennessy, A. Ingólfsdóttir, A theory of communicating processes with value-passing, Inf. Comput. 107 (1993) 202–236.
[28] M. Hennessy, H. Lin, Symbolic bisimulations, Theor. Comput. Sci. 138 (1995) 353–369.
[29] C. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
[30] S. Kleene, General recursive functions of natural numbers, Math. Ann. 112 (1936) 727–742.
[31] J. van Leeuwen, J. Wiedermann, A theory of interactive computation, in: Dina Goldin, Scott A. Smolka, Peter Wegner (Eds.), Interactive Computation: The New Paradigm, Springer, 2006, pp. 119–142.
[32] B. Luttik, F. Yang, The π-calculus is behaviourally complete and orbit-finitely executable, Log. Methods Comput. Sci. 17 (1) (2021) 14:1–14:26.
[33] R. Milner, Communication and Concurrency, Prentice Hall, 1989.
[34] R. Milner, Functions as processes, Math. Struct. Comput. Sci. 2 (1992) 119–146.
[35] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, Inf. Comput. 100 (1992) 1–40 (Part I); Inf. Comput. 100 (1992) 41–77 (Part II).

[36] R. Milner, D. Sangiorgi, Barbed bisimulation, in: ICALP'92, in: Lecture Notes in Computer Science, vol. 623, 1992, pp. 685–695.
[37] U. Nestmann, What is a good encoding of guarded choices?, Inf. Comput. 156 (2000) 287–319.
[38] U. Nestmann, Welcome to the jungle: a subjective guide to mobile process calculi, in: CONCUR'06, in: Lecture Notes in Computer Science, vol. 4137, 2006, pp. 52–63.
[39] U. Nestmann, B. Pierce, Decoding choice encodings, in: CONCUR'96, in: Lecture Notes in Computer Science, vol. 1119, 1996, pp. 179–194.
[40] F. Nielsen, The Typed λ-Calculus with First Class Processes, Report ID-TR:1988-43, Institute for Datateknik, Tekniske Hojskole, Denmark, Computer Science Department, University of Aarhus, 1986.
[41] M. Nielsen, I. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000.
[42] C. Palamidessi, Comparing the expressive power of the synchronous and the asynchronous π-calculus, Math. Struct. Comput. Sci. 13 (2003) 685–719.
[43] D. Park, Concurrency and automata on infinite sequences, in: Theoretical Computer Science, in: Lecture Notes in Computer Science, vol. 104, 1981, pp. 167–183.
[44] J. Parrow, D. Sangiorgi, Algebraic theories for name-passing calculi, Inf. Comput. 120 (1995) 174–197.
[45] C. Petri, Communication with Automata, Dissertation, Darmstadt Technical University, 1962.
[46] M. Presburger, Über die Vollständigkeit eines Gewissen Systems der Arithmetik Ganzer Zahlen, in welchem die addition als einzige operation hervortritt, in: Warsaw Mathematics Congress, vol. 395, 1929, pp. 92–101.
[47] L. Priese, On the concept of simulation in asynchronous, concurrent systems, Prog. Cybern. Syst. Res. 7 (1978) 85–92.
[48] H. Rogers, Theory of Recursive Functions and Effective Computability, MIT Press, 1987.
[49] D. Sangiorgi, Expressing Mobility in Process Algebras: First Order and Higher Order Paradigm, Ph.D. thesis, Department of Computer Science, University of Edinburgh, 1992.
[50] D. Sangiorgi, From π-calculus to higher order π-calculus – and back, in: Proc. TAPSOFT'93, in: Lecture Notes in Computer Science, vol. 668, 1993, pp. 151–166.
[51] D. Sangiorgi, D. Walker, The π Calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
[52] E. Santos, Computability by probabilistic Turing machines, Trans. Am. Math. Soc. 159 (1971) 165–184.
[53] R. Soare, Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets, Springer-Verlag, Heidelberg, 1987.
[54] B. Thomsen, A calculus of higher order communicating systems, in: Proc. POPL'89, 1989, pp. 143–154.
[55] B. Thomsen, A theory of higher order communicating systems, Inf. Comput. 116 (1995) 38–57.
[56] A. Turing, On computable numbers, with an application to the entsheidungsproblem, Proc. Lond. Math. Soc. 42 (1936) 230–265.
[57] J. Wiedermann, J. Leeuwen, How we think of computing today, in: Proceedings of the 4th Conference on Computability in Europe: Logic and Theory of Algorithms, Springer-Verlag, 2008, pp. 579–593.
[58] X. Xu, Distinguishing and relating higher-order and first-order processes by expressiveness, Acta Inform. 49 (2012) 445–484.