Thesis for Interaction

Yuxi Fu
BASICS, Shanghai Jiao Tong University
fu-yx@cs.sjtu.edu.cn

***Abstract*—A thesis for interaction model is proposed as a foundation for interactive solvability. It formulates the minimal content computability and the maximal channel computability of all interaction models 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, in the maximal model, of interpreters for the models that fit in the thesis.**

I. 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

Physical implementability is the very foundation of computer science. Church-Turing Thesis states that the functions implementable by physical devices are precisely those definable by the mathematical models of computation [12], [3], [36], [16]. 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 [22]. Leaving the implementation detail aside, it has been shown that such a model is equivalent to Turing machine model as far as computability is concerned. All experiments indicate that the world is quantum mechanical in that all physical phenomena can be explained by quantum mechanics. If we believe what physicists believe, then the unsolvability theory based on 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 [32] for example, believed by many to give a broader picture of efficient computation [10], is equivalent to Turing machine model in computability. Physical implementability emphasizes another property of computation, that of 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 observability rules out models that deal with general functions on the field of real number.

The physical implementability remains a fundamental constraint if one switches 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. What is the expressiveness of distributed computing? How about Internet computing? For closed systems of interactive processes 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 that 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

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

such that $i_1 < i_2 < \ldots$ and $\{i_1, i_2, \ldots\}$ is the set of 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 an input function is total. 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 Church-Turing Thesis there is nothing new about interaction models. What is new is that processes have access to channels. They can choose to communicate, or not to communicate at a particular channel. They can also send the identify of a channel to other processes. To what extent manipulations on channels should be allowed? It is this aspect of interaction that has not been under scrutiny in the light of 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 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. 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 need be discussed here. One is about the rational behind our design decision to take as atomic a communication of a number whose size is unbounded. Ob-

serve that this is consistent with the design of Turing machines. The input and the output tapes of a machine are unbounded in length. This is so because a machine is able to take inputs of any length and may output a string of any length. When we turn Turing machines into interactive machines, the input and output tapes become channels. 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? The answer is definitely negative. 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 packets. At the theoretical level a process calculus with a token-passing communication mechanism is not very much different from CCS, and runs the risk of not being a complete model [8]. It is at the right level of abstraction that a complete piece of information is transmitted in an atomic interaction. The second point is concerned with dynamic creation of private channels. Since we cannot restrict the number of global channels a process need 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? Take a look at the process $!a(x).(p)(p(y).S \mid b(z).\overline{p}[z].T)$. Every time an interaction with the process occurs, a new private channel p is generated. If p is a physical wire, it would mean that computation can cause the creation of a new physical wire. From a physical implementation point of view, the idea that a single process is always in need of a new private channel is debatable. Fortunately there is an alternative interpretation of private channel. In $(p)(p(y).S \mid b(z).\overline{p}[z].T)$ the scope operator (p) translates to saying that "it does *not* matter which channel $p(y).S$ and $\overline{p}[z].T$ are going to use to interact, neither of them is allowed to communicate with anybody else in the next step". This is the view we shall adopt in our universal model.

After fifty years of concurrency theory [25], we are still in a situation where we are expected to define everything from scratch. We cannot go very far that way. The lack of a definability/unsolvability theory of processes is the prime reason. Ultimately this is due to the absence of a fundamental thesis for theory of interaction. Our understanding of concurrency theory must reach to a stage when new results are built from known results and the whole theory is based upon a firm foundation. In this paper we attempt to postulate such a thesis with the motivation described in the above.

The rest of the paper is organized as follows. Section II defines the principal model of the paper, a functional model for concurrency. Section III reviews the preliminaries of the theory of interaction developed in [8], and apply them to the principal model. Section IV defines the minimal model and the maximal model and postulates a thesis for interaction models in terms of the two models. It is pointed out that the interactive versions of the recursive function model and the Turing machine models are admitted by the thesis. In Section V and Section VI it is proved that the functional model and the π-calculus fall within the reign of the thesis. A theory of definability/unsolvability is outlined in Section VII. Section VIII concludes.

II. A Concurrent Functional Model

Church's λ-calculus [1] is a prototypical higher order model for functional computation. It is of higher order in the sense that both an input and an output of a λ-term are themselves λ-terms. In concurrency theory an important issue has been to look for the model that plays the role of being the "λ-calculus" for concurrent computation. Early works [21], [2], [34], [35] addressed this issue by introducing the higher order mechanism into CCS [17]. A crucial shift from the functional paradigm to an object-oriented paradigm, initiated by the work of [4], was fully investigated by Milner, Parrow and Walker with the π-calculus [19]. The model is a channel-passing calculus. Our presentation of the model distinguishes syntactically private channels from global channels. The following notational convention will be maintained throughout the paper.

- C_g is the set of global channels. The elements of C_g are denoted by a, b, c, d, e, f, g, h.
- C_p is the set of private channels. The elements of C_p are denoted by l, m, n, o, p, q.
- C_v is the set of channel variables. The elements of C_g are denoted by u, v, w, x, y, z.

We write for example $\widetilde{p}$ for a finite sequence of private channels. The set $C_g \cup C_p \cup C_v$ will be ranged over by μ, ν, and the set $C_g \cup C_p$ by α, β.

We start by fixing the grammar of the minimal π-calculus.

$$T \quad := \quad \mathbf{0} \mid \mu(x).T \mid \mu\mu'.T \mid T \mid T' \mid (p)T \mid !\mu(x).T.$$

We will use the standard abbreviation $\mu(p).T \equiv (p)\mu p.T$. The labeled transition semantics is defined by the following rules.

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

$$\frac{T_0 \xrightarrow{\alpha\alpha'} T_0' \quad T_1 \xrightarrow{\overline{\alpha}\alpha'} T_1'}{T_0 \mid T_1 \xrightarrow{\tau} T_0' \mid T_1'} \qquad \frac{T_0 \xrightarrow{\alpha p} T_0' \quad T_1 \xrightarrow{\overline{\alpha}(p)} T_1'}{T_0 \mid T_1 \xrightarrow{\tau} (p)(T_0' \mid T_1')}$$

$$\frac{}{!\alpha(x).T \xrightarrow{\alpha\alpha'} T\{\alpha'/x\} \mid !\alpha(x).T}$$

$$\frac{T_0 \xrightarrow{\lambda} T_0'}{T_0 \mid T_1 \xrightarrow{\lambda} T_0' \mid T_1} \qquad \frac{T \xrightarrow{\lambda} T'}{(p)T \xrightarrow{\lambda} (p)T'} \; p \text{ is not in } \lambda.$$

The symmetric rules are systematically omitted. The last two rules are structural rules. They are common to all models and will be omitted in the definition of a new model. We shall use standard notations like $\Longrightarrow$, $\overset{\lambda}{\Longrightarrow}$ and $\xrightarrow{\lambda_1 \ldots \lambda_k}$.

The π-calculus has been successful for many reasons. It is powerful enough to simulate many process calculi and is capable of modelling a wide range of phenomena arising from concurrent programming [39], [29], [30], [37]. Milner's encoding of the lazy λ-calculus into the π-calculus [18] confirms that the object-oriented paradigm is powerful enough to encapsulate the functional computation. The relationship to the λ-calculus is seen as a qualification test for a model to

claim the "λ-calculus" status. The object-oriented style of π-programming is evident from the following encoding of the lazy λ-calculus given by Milner:

$$\begin{aligned} [\![x]\!]_u &= \overline{x}u, \\ [\![\lambda x.M]\!]_u &= u(x).u(v).[\![M]\!]_v, \\ [\![MN]\!]_u &= (p)([\![M]\!]_p \mid \overline{p}(w).\overline{p}u.!w(z).[\![M]\!]_z). \end{aligned}$$

Every λ-term in the π-calculus is accessible at some channel. A term is invoked only when it is needed, and the simulation is indirect. A natural question asks if there is a more direct interpretation of the λ-calculus in a concurrent model?

We propose in this section an alternative candidate for the "λ-calculus" of concurrent computation. It is a higher order *functional* model, denoted by $\mathbb{F}$, with a direct interpretation of the lazy λ-calculus. The following syntactical class is necessary for the definition of the model.

- $\mathcal{V}$ is the set of abstraction variables. The elements of C_g are denoted by U, V, W, X, Y, Z.

The set of $\mathbb{F}$-*terms* is generated by the following grammar:

$$\begin{aligned} T &:= \mathbf{0} \mid \gamma.T \mid T \mid T \mid (p)T \mid A(\mu, \nu), \\ A &:= X \mid (x, y)T, \ \textit{where } fc(T) \subseteq \{x, y\} \wedge oc(T) = \emptyset, \\ \gamma &:= \mu(X) \mid \overline{\mu}[A]. \end{aligned}$$

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. We write $fc(T)$ for the set of the free channel variables appearing in T. If y does not appear in S, 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 $A(\mu, \mu) = T\{\mu/x\}$. We will write $A_{\mu\nu}$ for $A(\mu, \nu)$ and A_μ for $A(\mu)$ to improve readability. The set of abstractions will be ranged over by A, B, C, D. The nil process $\mathbf{0}$ and the composition term $T \mid T'$ are standard. A trailing $\mathbf{0}$ is often omitted. A *prefix* term $\gamma.T$ is either an *input* term of the form $\mu(X).T$ or an *output* term of the form $\overline{\mu}[A].T$. The abstraction variable X in $\mu(X).T$ is *bound*. A (abstraction) variable is free if it is not bound. We write $fv(T)$ for the set of free variables appearing in T. We shall abbreviate $\mu(X).T$ to $\mu.T$ if X does not appear in T. Similarly we shall abbreviate $\overline{p}[A].T$ to $\overline{p}.T$ in a context where A can never be received by others. We will abbreviate for example $a(X).a(Y).T$ to $a(X, Y).T$ and $\overline{a}[A].\overline{a}[B].T$ to $\overline{a}[A, B].T$. We will call $a(X)$ a global input guard, $\overline{a}[A]$ a global output guard, $p(X)$ a local input guard, and $\overline{p}[A]$ a local output guard. A τ prefix term $\tau.t$ can be defined by $(p)(p.T \mid \overline{p})$ where $p \notin fc(T)$. In *localization* term $(p)T$ the private channel p is localized. A private channel is *open* if it is not localized. We write $oc(T)$ for the set of the open private channels appearing in T.

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

We try to keep the model as small as possible. The composition and localization are basic operations necessary in all interaction models. Prefix terms introduce sequential actions. We insist that all abstractions have two parameter variables. We think of an abstraction as a concurrent generalization of function. Two parameters are necessary and sufficient to specify the input and the output interface of a function. We believe that the model would be far less expressive if all abstractions have only one parameter, although currently we have not formally proved this. Working with fixed number parameter abstractions saves us from introducing a type system for channels. Abstraction-passing higher order calculi have been studied in literature [30], [37]. It is a more faithful extension of the higher order mechanism of the λ-calculus than strict process-passing calculi. Strict process-passing calculi are not expressive enough since a receiving process has no control over a received process [8]. One can enhance the expressive power of process-passing calculi by introducing relabeling operator. The problem with the relabeling operator is that it interferes with congruence property.

Using the action set $\{\alpha(A), \mid \alpha \in C_g \cup C_p\} \cup \{\tau\}$, where τ is a label for all internal actions, the simple operational semantics of $\mathbb{F}$ can be defined by the prefix rules

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

and the interaction rule

$$\frac{T_0 \xrightarrow{\alpha(A)} T_0' \qquad T_1 \xrightarrow{\overline{\alpha}(A)} T_1'}{T_0 \mid T_1 \xrightarrow{\tau} T_0' \mid T_1'}$$

The side condition on the grammar of abstraction simplifies the operational semantics. There is no name extrusion.

We need to make sure that the side condition on abstraction is maintained.

Lemma 1. *If T is an $\mathbb{F}$-term and $T \xrightarrow{\lambda} T'$, T' is an $\mathbb{F}$-term.*

Proof: Suppose $\lambda = a(A)$. If A lands in an output prefix, then speaking from that output position it can be instantiated by neither a free variable channel nor an open private channel. ∎

Let's see how recursion is defined in $\mathbb{F}$. The counterpart of the λ-term xx is $(p)(\overline{p}[X] \mid Xp)$. The infinite behaviour of $(\lambda x.xx)(\lambda x.xx)$ is simulated by the process

$$\Omega \stackrel{\text{def}}{=} (p)(\overline{p}[D] \mid Dp)$$

where the abstraction D is $(z)z(Z).(\overline{p}[X] \mid Xp)$. A term that inputs an abstraction at μ and then immediately outputs it at ν can be defined as $(p)(\overline{p}[C] \mid Cp))$, where $C = (z)\mu(X).\overline{\nu}[X].z(Z).(\overline{z}[Z] \mid Zz)$. This example can be generalized. We will use the following *guarded recursion* throughout the paper.

$$!\gamma_1 \ldots \gamma_k \dagger T \quad = \quad \gamma_1 \ldots \gamma_k.(T \mid (p)(\overline{p}[C] \mid Cp)), \qquad (2)$$

where $C = (z)\gamma_1 \ldots \gamma_k.z(Z).(T \mid \overline{z}[Z] \mid Zz)$ and p does not appear in T. Intuitively $!\gamma_1 \ldots \gamma_k \dagger T$ is a recursively defined procedure, each recursive call of the body T must be preceded by actions prescribed by $\gamma_1, \ldots, \gamma_k$. We will abbreviate $!\gamma_1 \ldots \gamma_k \dagger \mathbf{0}$ to $!\gamma_1 \ldots \gamma_k$.

III. Equality and Submodel

Milner-Park's bisimulation approach [17], [24] to observational equivalence has been a standard in concurrency theory. Two refinements of the idea have played an important role in recent studies of interaction models. The branching bisimulation of van Glabbeek and Weijland [38] imposes the condition that a change-of-state internal action must be bisimulated. The barbed bisimulation of Milner and Sangiorgi [20] provides for the first time a model independent equivalence for processes. Central to barbed bisimulation is the requirement that it is closed under composition and that it preserves observability.

Definition 2. *P is* observable at *a, notation $P\Downarrow_a$, if $P \xrightarrow{\lambda} P'$ for some P' and some action λ at channel a.*

In addition to the bisimulation property we shall pay particular attention to divergence. Our translations between models are supposed to be divergence sensitive. The criterion we adopt for divergence sensitivity was introduced by Priese [27].

Definition 3. *A symmetric relation $\mathcal{R}$ on processes is a* bisimulation *if one of the following is valid whenever $Q\,\mathcal{R}\,P \xrightarrow{\lambda} P'$:*

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

It is codivergent *if $P\,\mathcal{R}\,Q \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} Q_i \xrightarrow{\tau} \cdots$ implies that $P \overset{\tau}{\Longrightarrow} P'\,\mathcal{R}\,Q_k$ for some $k \geq 1$ and some process P'. It is* extensional *if $(P \mid P')\,\mathcal{R}\,(Q \mid Q')$ 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 all $a \in C_g$.*

The four properties defined in Definition 3 apply to all models. This is because they do not refer to any particularity of external actions. Internal actions are basically the same in all models [7] and is therefore model independent.

A. Process Equality

Putting together all the conditions introduced in Definition 3, we get the process equality we shall use in this paper.

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

It is a standard exercise to show that the largest reflexive, codivergent, extensional and equipollent bisimulation exists [8], hence the well-definedness of $=_{\mathbb{M}}$. Notice that we need the reflexivity for the proof of the existence of $=_{\mathbb{M}}$. Without the reflexivity the binary relation $\{(!\tau \mid !a, !\tau \mid !\overline{a}) \mid a \in C_g\}$ would be a codivergent, extensional, equipollent bisimulation. We shall often omit the subscript in $=_{\mathbb{M}}$. The equality $=_{\mathbb{M}}$ is introduced in [8], based on previous studies on the relative expressiveness of CCS variants and π variants [9]. In [8] the private channels and the global channels are syntactically the same, following the standard practice in process algebra [17], [19], [31]. Consequently the extensionality condition is supposed to be closed under localization operator. In the present paper this is unnecessary since all private channels in processes are bound. The effectiveness of the equality $=_{\mathbb{M}}$ has been demonstrated by the proof of a number of general results. See [8] for a systematic study.

A number of equivalences for higher order calculi have been proposed and studied, notably by Thompsen [34], [35] and Sangiorgi [29], [30]. These are weak bisimilarities without any consideration to divergence. Divergence is important for us since we shall discuss divergence sensitive interpretation between models. Divergence is also important for the interpretation of computational models, say the lazy λ-calculus.

Proving that two processes are absolutely equal is often not so straightforward. In many places of this paper we can use the stronger but much easier equality defined next.

Definition 5. *A symmetric codivergent bisimulation $\mathcal{R}$ is a T-bisimulation if the following hold whenever $P\mathcal{R}Q$:*

1) *If $P \xrightarrow{a(A)} P'$, then $Q \Longrightarrow Q'' \xrightarrow{a(A)} Q'$ for some Q', Q'' such that $Q''\mathcal{R}P$ and $Q'\mathcal{R}P'$.*
2) *If $P \xrightarrow{\overline{a}(A)} P'$, then $Q \Longrightarrow Q'' \xrightarrow{\overline{a}(B)} Q'$ for some Q', Q'' such that $Q''\mathcal{R}P$ and $Q'\mathcal{R}P'$ and $A(a,b)\,\mathcal{R}\,B(a,b)$ for all $a, b \in C_g$.*

The T-bisimilarity $\simeq_T$ is the largest T-bisimulation.

The proof that $\simeq_T$ is included in $=_{\mathbb{F}}$ is not difficult.

B. Direct Interpretation of λ-Calculus

The syntax of the λ-calculus is given by

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

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 of the lazy λ-calculus is defined by the following two rules:

$$\begin{array}{rcl} (\lambda x.M)N & \to & M\{N/x\}, \\ MN & \to & M'N, \quad \text{if } M \to M'. \end{array}$$

The equivalence closure of the one step lazy reduction $\to$ is the β-conversion $=_\beta$. The lazy reduction of a λ-term is unique. It either terminates in an abstraction term, or terminates in a term of the form $xM_1 \ldots M_k$, or diverges.

Given an injective function from the set of λ variables to the set $\mathcal{V}$ of abstraction variables, a λ-term M is interpreted as an abstraction according to the following definition.

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

The process $[\![M]\!]_{a,b}$ is a "function" that inputs a "λ-term" at channel a and output the result "λ-term" at channel b. The correctness of the interpretation is guaranteed by the following lemmas and corollary. We omit the routine proofs.

Lemma 6. *If $[\![x]\!] = X$ then $[\![M]\!]\{[\![N]\!]/X\} = [\![M\{N/x\}]\!]$.*

Lemma 7. *$M \to N$ if and only if $[\![M]\!]_{a,b} \to\!\!\to =_{\mathbb{F}} [\![N]\!]_{a,b}$.*

Corollary 8. *For closed M, N, $M =_\beta N$ implies $[\![M]\!] =_{\mathbb{F}} [\![N]\!]$.*

C. Relative Expressiveness

Another fundamental relationship in computer science is the relative expressiveness between models. For such a relationship to be useful at all it must be defined in a model independent way. Intuitively we regard $\mathbb{M}$ as a submodel of $\mathbb{N}$ if for every process P of the former there is some process Q of the latter such that Q is equal to P as it were; and moreover the equality $=_{\mathbb{M}}$ should be a subrelation of the equality $=_{\mathbb{N}}$. At the technical level the definition of the expressiveness should be a routine once the definition of equality has been fixed. The reflexivity condition of Definition 4 must be generalized to conditions on processes from two different models.

Definition 9. *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 $P \propto Q$, $P' \propto Q'$ and $P =_{\mathbb{M}} P'$ imply $Q =_{\mathbb{N}} Q'$.*

The soundness condition refers to both the equality of the source model $\mathbb{M}$ and the equality of the target model $\mathbb{N}$. For the condition to make the best sense of it the two equalities ought to be the same as it were. In our situation they are the same equality on two different models. The basic idea of the next definition is that the expressiveness relation is also the same equality, albeit on the processes of two models.

Definition 10. *A binary relation $\propto$ from $\mathbb{M}$-processes to $\mathbb{N}$-processes is a* subbisimilarity *if it is a total, sound, codivergent, extensional, equipollent bisimulation.*

We write $\mathbb{M} \sqsubseteq \mathbb{N}$ if there is a subbisimilarity from $\mathbb{M}$ to $\mathbb{N}$. The notation $\mathbb{M} \sqsubseteq \mathbb{N}$ stands for the formal statement that $\mathbb{M}$ is a submodel of $\mathbb{N}$. For example it is easy to see that $\mathbf{CCS} \sqsubseteq \pi$. We write $\mathbb{M} \sqsubset \mathbb{N}$ if $\mathbb{M} \sqsubseteq \mathbb{N}$ and $\mathbb{N} \not\sqsubseteq \mathbb{M}$. In [8] the subbisimilarity relationship has been used to classify the relative expressiveness of several well-known models. Our criteria are stronger than most criteria proposed in literature [23], [11].

IV. Thesis on Interaction

With the requirement on physical implementability comes the spatial operator "|", the concurrent composition operator, that allows processes to interact on shared channels. This is a let-it-happen combinator to which we do not and should not assign any extra computational meaning. The semantics of this operator is fixed; it is model independent. In literature one sees several variants of this combinator. We take the view that there variants are all derived operators. This will be formalized in our thesis. Without any hierarchical structure in space the spatial operator would be much less useful, hence the private channels and the localization operator. The localization operator is a scope operator without any extra computational meaning either. Its semantics is essentially model independent.

The other operators of an interaction model are intensional. They define computational features pertain to particular models. An interaction model differs from another interaction model in that they have different set of intensional operators. These are the principles for the introduction of the absolute equality and the subbisimilarity.

Following these principles it is easy to extend a computation model to an interaction model. Suppose we would like to define interactive Turing machine model. We let each machine to have the capacity to input and output data at a finite number channels. There could be many variations on the choice of instruction set. But the model is basically the same. In this model we can form systems by composing individual machines. For instance we have a system or a process of the form $(p)(M_0 \,|\, (q)(M_1 \,|\, M_2))$ where M_0, M_1, M_2 are interactive machines. The particularity of this model is that the configuration of a system never changes. For a particular formalization of the model, the reader is referred to [8]. We shall denote this model by $\mathbb{T}$.

The contents of communications in $\mathbb{T}$ are numbers. Formally we use Presburger Arithmetic to reason about numbers. Presburger Arithmetic [26] 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 always abbreviate say "((0 + 1) + 1) + 1" to "3". 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 [26]. Throughout the paper i, j, k stand for natural numbers.

A. Computability 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 denoted by $\mathbb{C}$. The $\mathbb{C}$-processes are generated from the following grammar.

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

The process Ω can only diverge. The functional process $F_a^b(\mathsf{f}(x))$ inputs a number, say i, at channel a, carries out the computation of $\mathsf{f}(i)$, and outputs the result 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 f is undefined on i.

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

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

$$\frac{P \xrightarrow{a(i)} P' \qquad Q \xrightarrow{\overline{a}(i)} Q'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, Q'} \tag{3}$$

Definition 4 can be readily applied to the $\mathbb{C}$-processes. The equality $=_{\mathbb{C}}$ has a simple characterization. Let the structural congruence $\equiv_{\mathbb{C}}$ be the least equivalent and congruent relation that satisfies the following equalities: (i) $\mathbf{0} \,|\, P \equiv_{\mathbb{C}} P$, (ii) $P \,|\, Q \equiv_{\mathbb{C}} Q \,|\, P$, (iii) $O \,|\, (P \,|\, Q) \equiv_{\mathbb{C}} (O \,|\, P) \,|\, Q$, (iv) $\Omega \,|\, \Omega \equiv_{\mathbb{C}} \Omega$, and (v) $F_a^b(\mathsf{f}(x)) = F_a^{b'}(\mathsf{f}(x))$ if $\mathsf{f}(x)$ is nowhere defined. The following lemma is from [8].

Lemma 11. *The two equalities $=_{\mathbb{C}}$ and $\equiv_{\mathbb{C}}$ coincide.*

B. Universal Model

The model $\mathbb{C}$ introduces a minimal expressive power on the interaction models, the *content computability*. In the other direction our universal interaction model imposes a constraint on all interaction models, which we call *channel computability*. The model is an extension of the value-passing calculus $\mathbb{VPC}$ of Hoare [15] and Milner [17]. To understand the role of $\mathbb{VPC}$ it suffices to say that $\mathbb{VPC}$ is to the recursion theory what $\mathbb{F}$ is to the λ-calculus. The syntax of the model is given by

$$T \quad := \quad \mathbf{0} \mid \alpha(x).T \mid \overline{\alpha}(t).T \mid T \mid T \mid (p)T \mid \textit{if } \varphi \textit{ then } T \mid !\alpha(x).T.$$

In the conditional term *if* φ *then* T the expression φ is a Presburger formula. The replication operator "!" only applies to input terms. We may define $!\overline{\alpha}(x).T$ by $(p)(\overline{p} \mid !p.\overline{\alpha}(x).\overline{p}.T)$.

The semantics is standard. In addition to the concurrent composition rules defined in (3) there are following rules.

$$\frac{}{\alpha(x).T \xrightarrow{\alpha(i)} T\{i/x\}} \qquad \frac{T \xrightarrow{\lambda} T' \quad \vdash \varphi}{\textit{if } \varphi \textit{ then } T \xrightarrow{\lambda} T'}$$

$$\frac{\vdash t = i}{\overline{\alpha}(t).T \xrightarrow{\overline{\alpha}(i)} T} \qquad \frac{}{!\alpha(x).T \xrightarrow{\alpha(i)} T\{i/x\} \mid !\alpha(x).T}$$

A detailed exposure of $\mathbb{VPC}$ can be found in [6].

The universal model $\mathbb{V}$ differs from $\mathbb{VPC}$ in that the former has a naming function $\mu : \mathbf{N} \to C_g$ that assigns a distinct number to each global channel. For a number i we will write μ_i for $\mu(i)$. The bijective function is meant to capture the intuition that global channels can and must be accessed in a computable way. The function μ gives rise to an enumeration $\mu_0, \mu_1, \ldots$ of the global channels. When we write a global channel a for example we mean that $a = \mu_k$ for some number k. The syntax of $\mathbb{V}$ is almost the same as $\mathbb{VPC}$, except that the former has additional prefix terms of the form $\mu_z(x).T$ and $\mu_z(t).T$. The semantics of $\mathbb{V}$ is the same as that of $\mathbb{VPC}$. The process $a(x).\textit{if } \textit{decode}(x) = k \textit{ then } \overline{\mu_k}(t)$ for instance decodes the number received at channel a and if the decoded number is the key then it outputs t at the secret channel. 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.

$$\begin{aligned} \mu_s(x).T &= (p)(\overline{p}(s) \mid p(z).\mu_z(x).T), \\ \overline{\mu_s}(t).T &= (p)(\overline{p}(s) \mid p(z).\overline{\mu_z}(t).T). \end{aligned}$$

In $\mathbb{V}$ there is a maximal process $\mathbf{1}$ as it were defined by $\mathbf{I} \mid \mathbf{O}$, where

$$\begin{aligned} \mathbf{I} &= (p)(\overline{p}(0) \mid !p(x).p(x+1) \mid p(y).(\overline{p}(0) \mid \mu_y(z))), \\ \mathbf{O} &= (pq)(\overline{p}(0) \mid !p(x).p(x+1) \mid \overline{q}(0) \mid !q(x).q(x+1) \\ &\qquad \mid p(y).q(z).(\overline{p}(0) \mid \overline{q}(0) \mid \overline{\mu_y}(z))). \end{aligned}$$

Intuitively $\mathbf{1}$ is the process that can do any input action and any output action in a perpetual fashion.

The two-leg conditional term *if* φ *then* S *else* T is defined by *if* φ *then* $S \mid$ *if* $\neg\varphi$ *then* T. For clarity the term

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

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

C. Models of Interaction

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 it out without further ado.

Thesis on 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 absolute equality and the subbisimilarity relationship. One may cast doubt on the thesis by pointing out that there are other model independent equalities and submodel relationships. Does that mean that there are many theses for interaction? Our answer has to be that for mathematical models of interaction the equality and the submodel relationship are unique for the reason that the observability of numbers is a model independent notion. The definition of $=$, and that of $\sqsubseteq$ as well, is discovered; it is not invented. This remark also applies to Church-Turing Thesis. There has been no dispute about the extensional equality of function to which Church-Turing Thesis refers.

We call $\mathbb{M}$ *complete* if $\mathbb{C} \sqsubseteq \mathbb{M}$ and *sound* if $\mathbb{M} \sqsubseteq \mathbb{V}$. Intuitively a complete model is Turing powerful, and a complete and sound model is Turing equivalent, if one ignores the interactional aspect of the model. The model $\mathbb{C}$ is the starting point for the theory of definability of $\mathbb{M}$, and $\mathbb{V}$ provides the theory of unsolvability for all interaction models.

In [8] we established the completeness of $\mathbb{T}$ and $\mathbb{VPC}$, exploiting the simplification offered by Lemma 11.

Theorem 12. $\mathbb{C} \sqsubseteq \mathbb{T} \sqsubseteq \mathbb{V}$ *and* $\mathbb{C} \sqsubseteq \mathbb{VPC} \sqsubseteq \mathbb{V}$.

Proof: The embedding function from $\mathbb{VPC}$ to $\mathbb{V}$ witnesses $\mathbb{VPC} \sqsubseteq \mathbb{V}$. The validity of $\mathbb{T} \sqsubseteq \mathbb{V}$ is due to the fact that Turing machines can be interpreted by recursive functions. ■

We will prove in the following section that the functional model $\mathbb{F}$ is also complete and sound. So our most influential computation models can all be lifted to interaction models. We will also show that the π-calculus satisfies the Thesis on Interaction.

V. Completeness of Functional Model

We prove in this section that $\mathbb{F}$ is complete. The proof is typical of the completeness proofs. We shall explain it in some detail. The reader should pay attention to the role played by Lemma 11.

The first step to define the computable functions in $\mathbb{F}$ is to fix an encoding in $\mathbb{F}$ of the natural numbers. Let A^+ abbreviate $(xy)\overline{x}[A]$. The numbers are defined by the following abstractions:

$$\begin{aligned} \widehat{0} &= (xy)\overline{y}, \\ \widehat{k+1} &= (\widehat{k})^+. \end{aligned}$$

If we try to code up the recursive functions in $\mathbb{F}$, we soon realize that we need to define in $\mathbb{F}$ the conditional term

$$\textit{if } \varphi \textit{ then } T \textit{ else } T', \tag{4}$$

where the validity of φ is decidable. This is where the Presburger Arithmetic comes into the picture.

A. Definability of Presburger Condition

In this subsection we look at the case where φ is a Presburger formula containing no occurrences of quantifier. In other words, φ is either an atomic formula of the form $s = t$, or an atomic formula of the form $s < t$, or $\phi \wedge \psi$, or $\phi \vee \psi$ or $\neg\phi$ inductively. The general definition of (4) in which φ is a decidable predicate will follow from Theorem 15.

The process *if* $\varphi \wedge \varphi'$ *then* T *else* T' can be defined by

$$\textit{if } \varphi \textit{ then } (\textit{if } \varphi' \textit{ then } T \textit{ else } T') \textit{ else } T'.$$

Similarly *if* $\varphi \vee \varphi'$ *then* T *else* T' and *if* $\neg\varphi$ *then* T *else* T' can be defined by conditional processes involving simpler boolean formulae. It is therefore sufficient to define (4) for atomic formulae. Firstly we define the auxiliary process *if* $A = B$ *then* T *else* T' by

$$(lm)(\bar{l}[A].\overline{m}[B] \mid !l(X).m(Y)\dagger(nopq)T^{?}).$$

where $T^{?}$ stands for

$$\left(Xno \mid Ypq \mid n(U).(p(V).\bar{l}[U].\overline{m}[V] \mid q.T') \mid o.(p(V).T' \mid q.T)\right).$$

So if A and B are abstractions encoding the same number, then *if* $A = B$ *then* T *else* T' is equal to T, otherwise it is equal to T'. Secondly we interpret Presburger arithmetic term t as $[\![t]\!]_o$ which is supposed to release the value of t at channel o. The structural definition of $[\![t]\!]_o$ is given in Fig. 1. To define $[\![x]\!]_o$ we fix a bijective correspondence between the term variables of the Presburger Arithmetic and the abstraction variables. We assume that the capital X corresponds to the small x. Finally we define the process *if* $s = t$ *then* T *else* T' by

$$(pq)([\![s]\!]_p \mid [\![t]\!]_q \mid p(X).q(Y).\textit{if } X = Y \textit{ then } T \textit{ else } T').$$

The conditional process *if* $s < t$ *then* T *else* T' is defined in similar fashion.

B. Interpretability of Recursive Functions

With the help of the conditional processes, we are now able to fix an interpretation of the recursive functions. To start with we define *copy process* $C_{\mu\nu}$ by the following process

$$\mu(X).(mno)\left(\overline{m}[\widehat{0}] \mid !o(X)\dagger\overline{m}[X] \mid (l)(\bar{l}[X] \mid C)\right),$$

where

$$C = !l(X)\dagger(pq)(Xpq \mid m(Z).(p(Y).\bar{o}[Z^{+}].\bar{l}[Y] \mid q.\bar{\nu}[Z])). \quad (5)$$

The process C_{ab} intends to input an abstraction, which is supposed to be a number, at channel a and to output the number at channel b. If the imported number is corrupted, the copy process either terminates or interprets the imported abstraction as some number and outputs it. It is typically used as a prefix operation. Let the notation $C_{\mu\nu}.T$ stand for the process obtained from $C_{\mu\nu}$ by substituting $q.(T \mid \overline{\nu}[X])$ for $q.\overline{\nu}[X]$ in (5). The role of $C_{\mu\nu}$ in $C_{\mu\nu}.T$ is to prevent malicious code from ever getting into T.

We define a map $[\![\_]\!]_c$ that interprets every recursive function by an abstraction. Constant function $\lambda x_1 \ldots \lambda x_k.i$, successor function $\lambda x.x{+}1$ and projection function $\lambda x_1 \ldots \lambda x_k.x_i$ are interpreted respectively by the following abstractions.

$$\begin{aligned} Cn^i_k &= (uv)(p_1 \ldots p_k)(C_{up_1} \ldots C_{up_k} \mid p_i(X).\overline{\nu}[\widehat{i}]),\\ Sr &= (uv)(p)(C_{up} \mid p(X).\overline{\nu}[X^{+}]),\\ Pj^i_k &= (uv)(p_1 \ldots p_k)(C_{up_1} \ldots C_{up_k} \mid p_i(X).\overline{\nu}[X]). \end{aligned}$$

Suppose $G^0 = [\![\mathsf{g}_0]\!]_c$ for an i-ary recursive function g_0 and $G^1 = [\![\mathsf{g}_1]\!]_c, \ldots, G^i = [\![\mathsf{g}_i]\!]_c$ for some k-ary recursive functions $\mathsf{g}_1, \ldots, \mathsf{g}_i$. Let

$$T_j = p_1(X_1) \ldots p_k(X_k).(\overline{p_1}[X_1] \ldots \overline{p_k}[X_k] \mid G^j(X_1, \ldots, X_i)),$$

where $G^j(X_1, \ldots, X_k) = (q)(\overline{q}[X_1] \ldots \overline{q}[X_k] \mid G^j_{qo_i})$. The term T_j inputs the numbers at channels $p_1, \ldots, p_k$ consecutively and then must release these numbers in the same order so that they can be used elsewhere. Let $CP_{G^0,G^1,\ldots,G^i}$ be defined by

$$(o_1 \ldots o_i)(T_1 \mid \ldots \mid T_i \mid o_1(Z_1) \ldots o_i(Z_i).(n)(\overline{n}[Z_1] \ldots \overline{n}[Z_i] \mid G^0_{nv})).$$

The interpretation $[\![\mathsf{g}_0(\mathsf{g}_1, \ldots, \mathsf{g}_i)]\!]_c$ of the composition function $\mathsf{g}_0(\mathsf{g}_1, \ldots, \mathsf{g}_i)$ is defined by

$$(uv)(p_1 \ldots p_k)(C_{up_1} \ldots C_{up_k} \mid CP_{G^0,G^1,\ldots,G^i}).$$

Next we consider recursion and minimization. Let g, h, f be respectively a $(k{+}2)$-ary recursive function, a k-ary recursive function, and a $(k{+}1)$-ary recursive function, and let G, H, F be their interpretations. The recursion function $\mathsf{rec}(\mathsf{g}, \mathsf{h})$ defined from g and h is calculated by the following equations:

$$\begin{aligned} \mathsf{rec}(\mathsf{g}, \mathsf{h})(i_1, .., i_k, 0) &= \mathsf{h}(i_1, .., i_k),\\ \mathsf{rec}(\mathsf{g}, \mathsf{h})(i_1, .., i_k, i+1) &= \mathsf{g}(i_1, .., i_k, i, \mathsf{rec}(\mathsf{g}, \mathsf{h})(i_1, .., i_k, i)). \end{aligned}$$

The interpretation $[\![\mathsf{rec}(\mathsf{g}, \mathsf{h})]\!]_c$ defined in Fig. 2 uses an inductive algorithm. In the recursive call of $!o(Y, Z, V)\dagger$ $(\_)$, the parameter records the intermediate value $\mathsf{rec}(\mathsf{g}, \mathsf{h})(i_1, \ldots, i_k, y)$; Z is the $(k{+}1)$-th input; Y is y when the algorithm calculates $\mathsf{rec}(\mathsf{g}, \mathsf{h})(i_1, \ldots, i_k, y)$. The inductive algorithm terminates when Y reaches the $(k{+}1)$-th input value. The interpretation of the minimization function $\mathsf{mu}(\mathsf{f})$, given in Fig. 2, makes use of the standard algorithm. The definition of all recursive functions is now complete.

The minimization function introduces undefinedness. Our interpretation treats undefinedness as divergence. The soundness of our interpretation is the following lemma.

Lemma 13. *Suppose* f *is a* k*-ary recursive function. For all* $i_1, \ldots, i_k$, $(p)(\overline{p}[\widehat{i_1}, \ldots, \widehat{i_k}] \mid [\![f]\!]_c(p, \nu)) =_{\mathbb{F}} \overline{\nu}[\widehat{j}]$ *if* $\mathsf{f}(i_1, \ldots, i_k) = j$, *and* $(p)(\overline{p}[\widehat{i_1}, \ldots, \widehat{i_k}] \mid [\![f]\!]_c(p, \nu)) =_{\mathbb{F}} \Omega$ *if* $\mathsf{f}(i_1, \ldots, i_k)\uparrow$.

Proof: The notation Ω stands for the divergent process $(p)(\overline{p} \mid !p\dagger\overline{p})$. The proof is by simple structural induction. The internal manipulations are all state preserving. ∎

$$
\begin{aligned}
[\![k]\!]_o &= \overline{o}(\widehat{k}],\\
[\![x]\!]_o &= (pq)(Xpq \mid (p(U).\overline{o}[U^+] \mid q.\overline{o}[\widehat{0}])),\\
[\![t{+}1]\!]_o &= (p)([\![t]\!]_p \mid p(U).\overline{o}[U^+]),\\
[\![s{+}t]\!]_o &= (pq)([\![s]\!]_p \mid [\![t]\!]_q \mid p(X).q(Y).(ln)(\bar{l}[Y].\overline{n}[X] \mid !l(V)\dagger(pq)(Vpq \mid (p(U).n(Z).\bar{l}[U].n[Z^+] \mid q.n(W).\overline{o}[W])))).
\end{aligned}
$$

Fig. 1. Encoding of Presburger Terms.

$$
\begin{aligned}
[\![\mathsf{rec(g,h)}]\!]_c &= (uv)(p_1 \dots p_k)(C_{up_1} \dots C_{up_k}.C_{up} \mid (no)p_1(X_1) \dots p_k(X_k).p(Z).(\overline{n}[\widetilde{X}] \mid \overline{o}[\widehat{0}, Z, \widehat{0}]\\
&\quad \mid !o(Y,Z,V)\dagger \textit{ if } Y = \widehat{0} \wedge Z = \widehat{0} \textit{ then } H_{nv}\\
&\qquad \textit{else if } Y = \widehat{0} \wedge \widehat{0} < Z \textit{ then } (l)(H_{nl} \mid l(V).\overline{o}[\widehat{1}, Z, V])\\
&\qquad \textit{else if } Y > \widehat{0} \wedge Y < Z \textit{ then } n(\widetilde{X}).(\overline{n}[\widetilde{X}] \mid (lq)(\overline{q}[\widetilde{X}, Y, V] \mid G_{ql} \mid l(V).\overline{o}[Y^+, Z, V]))\\
&\qquad \textit{else if } Y > \widehat{0} \wedge Y = Z \textit{ then } n(\widetilde{X}).(\overline{n}[\widetilde{X}] \mid (q)(\overline{q}[\widetilde{X}, Y, V] \mid G_{qv})).\\
[\![\mathsf{mu(f)}]\!]_c &= (uv)(p_1 \dots p_k)(C_{up_1} \dots C_{up_k}.C_{up} \mid (no)p_1(X_1) \dots p_k(X_k).p(Z).(\overline{n}[\widetilde{X}] \mid \overline{o}[\widehat{0}]\\
&\quad \mid !o(Z)\dagger\ n(\widetilde{X}).(\overline{n}[\widetilde{X}] \mid (l)(\bar{l}[\widetilde{X}, Z] \mid F_{lq} \mid q(V).\textit{if } V = \widehat{0} \textit{ then } \overline{v}[Z] \textit{ else } \overline{o}[Z^+]))))).
\end{aligned}
$$

Fig. 2. Encoding of Recursion and Minimization Functions.

C. Completeness and Definability

We need another technical lemma for the main result of the section.

Lemma 14. *Suppose f, g are equivalent recursive processes. Then $[\![\mathsf{f}]\!]_c(ab) =_{\mathbb{F}} [\![\mathsf{g}]\!]_c(ab)$ for all a, b.*

Proof: In the light of Lemma 13, we only have to look at the case $[\![\mathsf{f}]\!]_c ab \xrightarrow{a(A)} L$ where A is not a number. This has been taken care of because the copy process will check if A is a number, and if not it will not proceed. And $[\![\mathsf{g}]\!]_c ab$ does the same. ∎

Every computable function corresponds to an infinite number of equal recursive functions. For each computable function $\mathsf{f}(x)$ let's fix a recursive function $\mathsf{f}_r(x)$ that computes it. We now define a straightforward encoding $[\![\_]\!]_\hbar$ of $\mathbb{C}$ into $\mathbb{F}$.

$$
\begin{aligned}
[\![\mathbf{0}]\!]_\hbar &= \mathbf{0},\\
[\![\Omega]\!]_\hbar &= \Omega,\\
[\![F^b_a(\mathsf{f})]\!]_\hbar &= [\![\mathsf{f}_r]\!]_c(ab),\\
[\![\overline{a}(i)]\!]_\hbar &= \overline{a}(\widehat{i}),\\
[\![P \mid Q]\!]_\hbar &= [\![P]\!]_\hbar \mid [\![Q]\!]_\hbar. \qquad (6)
\end{aligned}
$$

According to Lemma 13, the encoding is independent of the choice of the recursive functions. Let $\propto_\hbar$ be the composition of $[\![\_]\!]_\hbar$ followed by $=_{\mathbb{F}}$. More specifically

$$\propto_\hbar = \{(P, Q) \mid P \text{ is a } \mathbb{C} \text{ process, and } [\![P]\!]_\hbar =_{\mathbb{F}} Q\}.$$

It is a subbisimilarity from the $\mathbb{C}$-processes to the $\mathbb{F}$-processes.

Theorem 15. $\mathbb{C} \sqsubseteq \mathbb{F}$.

Proof: The total relation $\propto_\hbar$ is extensional by (6). It is an equipollent, codivergent bisimulation according to Lemma 13. Suppose $P =_{\mathbb{C}} Q$, $P \propto_\hbar P'$ and $Q \propto_\hbar Q'$. By Lemma 11, one has $P \equiv_{\mathbb{C}} Q$. It follows from the definition of $\equiv_{\mathbb{C}}$, (6), the definition of $\propto_\hbar$ and the equivalence property of $=_{\mathbb{F}}$ that $P' =_{\mathbb{F}} Q'$. ∎

VI. $\mathbb{V}$-Interpreter

This section serves two purposes. One is to show how to construct a $\mathbb{V}$-process that acts as an interpreter for $\mathbb{F}$. The other is to prove the soundness of $\mathbb{F}$ by exploiting the power of the interpreter. For the interpreter to work, there should be a way to translate an $\mathbb{F}$-process to a number, and vice versa. This is what Gödel encoding is about.

A. Gödel Index

For each k let $\langle \underbrace{\_, \dots, \_}_{k} \rangle : \underbrace{\mathbf{N} \times \dots \times \mathbf{N}}_{k} \to \mathbf{N}$ be an effective tupling function, where $\mathbf{N}$ denotes the set of natural numbers. The corresponding projection functions are denoted by $(\_)^k_0, \dots, (\_)^k_{k-1}$. We often write z_i for the j-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

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

The Gödel encoding is based on two bijective functions. The function $\varsigma : \mathcal{V} \to \mathbf{N}$ provides a countable enumeration $\{X_0, X_1, \dots\}$ of the abstraction variables. The bijective map

$$\sigma : C_v \cup C_g \cup C_p \to \mathbf{N}$$

is defined by

$$
\sigma(\mu) = \begin{cases} 3(i-1)+3, & \text{if } \mu = x_i,\\ 3i+1, & \text{if } \mu = \mu_i,\\ 3i+2, & \text{if } \mu = p_i, \end{cases} \qquad (7)
$$

assuming an enumeration $\{p_0, p_1, \dots\}$ of the private channels and an enumeration $\{x_0, x_1, \dots\}$ of the channel variables, and making use of the naming function μ.

The encoding function $[\![\_]\!]_{\mathsf{f}}$ for terms is defined below.

$$\begin{aligned}
[\![\mathbf{0}]\!]_{\mathsf{f}} &\stackrel{\text{def}}{=} 0,\\
[\![\mu(X).T]\!]_{\mathsf{f}} &\stackrel{\text{def}}{=} 5 * \langle \sigma(\mu), \varsigma(X), [\![T]\!]_{\mathsf{f}}\rangle + 1,\\
[\![\overline{\mu}[A].T]\!]_{\mathsf{f}} &\stackrel{\text{def}}{=} 5 * \langle \sigma(\mu), [\![A]\!]_{\mathsf{f}}, [\![T]\!]_{\mathsf{f}}\rangle + 2,\\
[\![T \mid T']\!]_{\mathsf{f}} &\stackrel{\text{def}}{=} 5 * \langle [\![T]\!]_{\mathsf{f}}, [\![T']\!]_{\mathsf{f}}\rangle + 3,\\
[\![(p)T]\!]_{\mathsf{f}} &\stackrel{\text{def}}{=} 5 * \langle \sigma(p), [\![T]\!]_{\mathsf{f}}\rangle + 4,\\
[\![A(\mu,\nu)]\!]_{\mathsf{f}} &\stackrel{\text{def}}{=} 5 * \langle \sigma(\mu), \sigma(\nu), [\![A]\!]_{\mathsf{f}}\rangle + 5,
\end{aligned}$$

and

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

Given an $\mathbb{F}$-term T, we call $[\![T]\!]_{\mathsf{f}}$ the *Gödel index* of T. Similarly we call $[\![A]\!]_{\mathsf{f}}$ 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 $[\![A]\!]_{\mathsf{f}}$ for any abstraction A, it is deemed as an index of $(x_0, x_1)\mathbf{0}$.

B. Interpreter

An *interpreter* $\mathcal{I}_\alpha$ *at* α inputs a number at channel α, interprets it as the Gödel index of some $\mathbb{F}$-process, and then simulates the $\mathbb{F}$-process. The process $\mathcal{I}_\alpha$ consists of a *parser* and a *simulator*. It is the process

$$\alpha(x).(l_i l_o l_1 l_0 m_i m_o m n o)\,(P_n(x) \mid S_n \mid \overline{m_i}(0) \mid \overline{m_o}(0) \mid \overline{m}(k_x) \mid L_i \mid L_o)\,.$$

The locks L_i, L_o are necessary to prevent the simulator from running into deadlock. The processes $\overline{m_i}(0), \overline{m_o}(0), \overline{m}(k_x)$ provide the initial values of three counters accessible at m_i, m_o, m respectively. The symbol k_x stands for the maximum of the indices of private channels appearing in x. The parser $P_n(x)$ parses a number imported at channel α to see if it is a Gödel index of some $\mathbb{F}$-process. It needs to check the following:

1) Are all channel variables bound?
2) Are all private channels localized?
3) Are all abstraction variables bound?

If the input number is the Gödel index of some $\mathbb{F}$-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 must be definable by the completeness of $\mathbb{F}$.

The simulator S_n defined in Fig. 3 receives the index at channel n and simulates the operations of P on-the-fly. For clarity the recursive definition of $In(y)$ and $Out(y)$ are defined recursively. They can be easily implemented in terms of the replication operator. The simulator makes use of the naming function to simulate transmissions at global channels. The simulation of an interaction at a private channel is a bit involved. Each process of the form $(q)T$ is translated to a $\mathbb{V}$-process, called a *dominion*, of the form

$$(o)\,(In(u) \mid Out(u) \mid \overline{n}\,([u/\varsigma(q)][\![T]\!]_{\mathsf{f}}))\,,$$

where $[u/\varsigma(q)][\![T]\!]_{\mathsf{f}}$ denotes the result obtained by substituting u for $\varsigma(q)$ in $[\![T]\!]_{\mathsf{f}}$. Different dominions make use of the same symbol o for private channels with *disjoint* scopes. In other words all private channels of an $\mathbb{F}$-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 choice. The component $\overline{n}([u/\varsigma(q)][\![T]\!]_{\mathsf{f}})$ invokes the simulator S_n recursively, which may take $[u/\varsigma(q)][\![T]\!]_{\mathsf{f}}$ out of the dominion it initially stays. The continuation of $[u/\varsigma(q)][\![T]\!]_{\mathsf{f}}$ typically wanders through different dominions. The question is then how to ensure that the private channel q appearing in T and its continuations is correctly interpreted. Our solution is to attach to every dominion a distinct number u. If a continuation of $[u/\varsigma(q)][\![T]\!]_{\mathsf{f}}$ tries to communicate at the private channel q, it must first of all move to the dominion where $\overline{n}([u/\varsigma(q)][\![T]\!]_{\mathsf{f}})$ originally stays before the action can be enabled. 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 $\mathbb{F}$-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 intends to input something at the global channel $\mu_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}$. If h is received at channel $\mu_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}$, 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 a channel 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 $\mu_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}$. 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 perfectly fine in an interleaving semantics. 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. 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_0 blocks $Out(u)$ with wrong u once it has been examined. That removes the possibility of generating infinite loops. The right $Out(\_)$ must be eventually reached. When that happens, the blocked $Out(\_)$'s are freed. 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. Had $Lock_o$ been released while some $Out(\_)$'s are blocked, it would be possible that a deadlock occurs. After the right dominion

$$
\begin{aligned}
S_n \;=\;& (pq)!n(z).\ \textbf{case}\ \mathsf{r}_5(z)\ \textbf{of}\\
& 1 \Rightarrow\ \textbf{case}\ \mathsf{r}_3(\mathsf{d}_5(z)_0)\ \textbf{of}\ 1 \Rightarrow \mu_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}(x).\overline{n}([x/\mathsf{d}_5(z)_1]\mathsf{d}_5(z)_2);\ 2 \Rightarrow \overline{l_i}.\overline{p}(z)\ \textbf{end case};\\
& 2 \Rightarrow\ \textbf{case}\ \mathsf{r}_3(\mathsf{d}_5(z)_0)\ \textbf{of}\ 1 \Rightarrow \overline{\mu_{\mathsf{d}_3(\mathsf{d}_5(z)_0)}}(\mathsf{d}_3(\mathsf{d}_5(z)_1)).\overline{n}(\mathsf{d}_5(z)_2);\ 2 \Rightarrow \overline{l_o}.\overline{q}(z)\ \textbf{end case};\\
& 3 \Rightarrow\ \overline{n}(\mathsf{d}_5(z)_0)\,|\,\overline{n}(\mathsf{d}_5(z)_1);\\
& 4 \Rightarrow\ m(u).\,(\overline{m}(u+3)\,|\,(o)(In(u)\,|\,Out(u)\,|\,\overline{n}([u/\mathsf{d}_5(z)_0]\mathsf{d}_5(z)_1)));\\
& 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);\\
& \textbf{end case};\\
In(y) \;=\;& p(z).\ \mathit{if}\ \mathsf{d}_5(z)_0 = y\ \mathit{then}\ m_i(v).\big(\mathit{if}\ v>0\ \mathit{then}\ \overline{l_1}.\overline{m_i}(v).In(y)\ \mathit{else}\ l_i.(I(z)\,|\,In(y))\big)\\
& \mathit{else}\ m_i(v).\overline{p}z.\overline{m_i}(v+1).l_1.m_i(v).\big(\mathit{if}\ v>1\ \mathit{then}\ \overline{l_1}.\overline{m_i}(v-1).In(y)\ \mathit{else}\ l_i.(I(z)\,|\,In(y))\big),\\
I(z) \;=\;& o(x).\overline{n}([x/\mathsf{d}_5(z)_1]\mathsf{d}_5(z)_2);\\
Out(y) \;=\;& q(z).\ \mathit{if}\ \mathsf{d}_5(z)_0 = y\ \mathit{then}\ m_o(v).\big(\mathit{if}\ v>0\ \mathit{then}\ \overline{l_0}.\overline{m_0}(v).Out(y)\ \mathit{else}\ l_o.(O(z)\,|\,Out(y))\big)\\
& \mathit{else}\ m_o(v).\overline{q}z.\overline{m_o}(v+1).l_0.m_o(v).\big(\mathit{if}\ v>1\ \mathit{then}\ \overline{l_0}.\overline{m_0}(v-1).Out(y)\ \mathit{else}\ l_o.(O(z)\,|\,Out(y))\big),\\
O(z) \;=\;& \overline{o}(\mathsf{d}_5(z)_1).\overline{n}(\mathsf{d}_5(z)_2).\\
L_i \;=\;& l_i.\overline{l_i}.L_i,\\
L_o \;=\;& l_o.\overline{l_o}.L_o;
\end{aligned}
$$

Fig. 3. Simulator S_n.

is found the simulator can simulate the output action by performing $\overline{o}(\mathsf{d}_5(z)_1)$ 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 gets a distinguished number.

This completes the explanation of the simulator.

C. Correctness

An output action say $\overline{a}[A]$ of an $\mathbb{F}$-process is bisimulated by the output action $\overline{a}(\llbracket A \rrbracket_{\mathsf{f}})$ of the interpretation. Once the deadlock prevention mechanism defined by $Out(y)$ and $In(y)$ is understood, the interpretation is easy to justify. It is sufficient to give an outline of how a subbisimilarity from $\mathbb{F}$ to $\mathbb{V}$ is constructed using the interpretation.

A local context $C[\ldots]$ is defined as follows:

1) A hole $[\_]$ is a local context;
2) $(p)C[\ldots]$ is a local context if $C[\ldots]$ is;
3) $C_0[\ldots]\,|\,C_1[\ldots]$ is a local context if $C_0[\ldots]$, $C_1[\ldots]$ are.

If $C[\ldots]$ is a local context with k holes, $C[T_1, \ldots, T_k]$ is the term obtained by filling the holes by terms $T_1, T_2, \ldots, T_k$. We define inductively a binary relation $\mathcal{R}$ between the set of $\mathbb{F}$-processes and the set of $\mathbb{V}$-processes in terms of local contexts. Starting from the empty set, we throw a pair (P, Q) into $\mathcal{R}$ if the following statements are valid:

1) $P = C[S_{k_1}, \ldots, S_{k_j}]$ for some local context $C[\_]$ in $\mathbb{F}$ and some $\mathbb{F}$-terms $S_{k_1}, \ldots, S_{k_j}$.
2) $Q = (l_i l_o l_1 l_0 m_i m_o mno)\,(S_n\,|\,\overline{m_i}(h)\,|\,\overline{m_o}(h)\,|\,\overline{m}(g)\,|\,L_i\,|\,L_o\,|\,O)$, where $O = (pq)D[T_{k_1}, \ldots, T_{k_j}]$, for some local context $D[\_]$ in $\mathbb{V}$ and some $\mathbb{V}$-terms $T_{k_1}, \ldots, T_{k_j}$.
3) $D[\ldots]$ is obtained from $C[\ldots]$ as follows: If $C[\ldots] = (q')C'[\ldots]$ then $D[\ldots] = (o)(In(k_{i'})\,|\,Out(k_{i'})\,|\,D'[\ldots])$ for some $k_{i'} \in \{i_1, \ldots, i_j\}$ and $D'[\ldots]$ is obtained from $C'[\ldots]$ in the same way. If $C[\ldots] = C_0[\ldots]\,|\,C_1[\ldots]$ then $D[\ldots] = D_0[\ldots]\,|\,D_1[\ldots]$ and $D_0[\ldots], D_1[\ldots]$ are obtained from $C_0[\ldots], C_1[\ldots]$ in the same way.
4) For each $k_{i'} \in \{k_1, \ldots, k_j\}$, $T_{k_{i'}} = \overline{n}[[k_{i'}/\sigma(q')]\llbracket S_{k_{i'}} \rrbracket_{\mathsf{f}}]$ with appropriate q'.

Now let $\mathcal{R}^{=}$ be the composition $\mathcal{R}; =_{\mathbb{V}}$. The relation $\mathcal{R}^{=}$ is equipollent because the capacity to perform actions at a particular global channel is preserved by the relation. 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 also extensional because $(\_)(\_\,|\,\overline{m_i}(h)\,|\,\overline{m_o}(h)\,|\,O)\,|\,(\_)(\_\,|\,\overline{m_i}(h')\,|\,\overline{m_o}(h')\,|\,O') = (\_)(\_\,|\,\overline{m_i}(h+h'+1)\,|\,\overline{m_o}(h+h'+1)\,|\,O'')$ for a suitable term O''. The soundness condition is also met because every number is the index of some abstraction. Hence the soundness of $\mathbb{F}$.

Theorem 16. $\mathbb{F} \sqsubseteq \mathbb{V}$.

Formally an interpretation $\llbracket \_ \rrbracket_{\mathbb{V}}$ of $\mathbb{F}$-processes in $\mathbb{V}$ is a function from the $\mathbb{F}$-processes to the $\mathbb{V}$-processes such that it is contained in a subbisimilarity from $\mathbb{F}$ to $\mathbb{V}$. By letting

$$\llbracket P \rrbracket_{\mathbb{V}} = (p)(\overline{p}[\llbracket P \rrbracket_{\mathsf{f}}]\,|\,\mathcal{I}_p),$$

we get an interpretation such that $P\mathcal{R}^{=}\llbracket P \rrbracket_{\mathbb{V}}$. In other words $\mathcal{I}_p$ is an interpreter of $\mathbb{F}$ at channel p.

D. Soundness of Mobile Calculus

The completeness of the π-calculus is proved in [8]. The soundness of the π-calculus can be established in similar way the soundness of $\mathbb{F}$ is proved.

Using the bijective function defined in (7), the Gödel index of a π-term is defined routinely as follows.

$$\begin{aligned}
[\![\mathbf{0}]\!]_\pi &\stackrel{\text{def}}{=} 0,\\
[\![\mu(x).T]\!]_\pi &\stackrel{\text{def}}{=} 5 * \langle \sigma(\mu), \sigma(x), [\![T]\!]_\pi \rangle + 1,\\
[\![\overline{\mu}\mu'.T]\!]_\pi &\stackrel{\text{def}}{=} 5 * \langle \sigma(\mu), \sigma(\mu'), [\![T]\!]_\pi \rangle + 2,\\
[\![T \mid T']\!]_\pi &\stackrel{\text{def}}{=} 5 * \langle [\![T]\!]_\pi, [\![T']\!]_\pi \rangle + 3,\\
[\![(p)T]\!]_\pi &\stackrel{\text{def}}{=} 5 * \langle \sigma(p), [\![T]\!]_\pi \rangle + 4,\\
[\![!\mu(x).T]\!]_\pi &\stackrel{\text{def}}{=} 5 * \langle \sigma(\mu), \sigma(x), [\![T]\!]_\pi \rangle + 5.
\end{aligned}$$

Using this Gödel encoding a $\mathbb{V}$-interpreter of the π-calculus can be constructed by basically recycling the idea of the algorithm defined in Fig. 3. The so called name extrusion does not cause any problem. Hence the following.

Theorem 17. $\mathbb{C} \sqsubseteq \pi \sqsubseteq \mathbb{V}$.

VII. Towards a Theory of Definability

The existence and the power of universal processes were investigated in [5]. The study was carried out in $\mathbb{VPC}$ with parametric definition. For the universal model $\mathbb{V}$ we can do much better.

Definition 18. *Fix a Gödel encoding of $\mathbb{V}$-terms, a universal process $\mathcal{U}_a$ for $\mathbb{V}$ at a is a $\mathbb{V}$-process such that (i) $\mathcal{U}_a$ can only perform input actions at a; and (ii) whenever $\mathcal{U}_a \stackrel{a(i)}{\longrightarrow} P$, then $P =_{\mathbb{V}} P_i$, where P_i is the $\mathbb{V}$-process with Gödel index i.*

The uniqueness of the universal process is an obvious corollary of the definition. Its existence can be demonstrated using the techniques developed in Section VI.

Theorem 19. *There is a unique universal process at a.*

Proof: If $\mathcal{U}_a$ and $\mathcal{U}'_a$ are two universal processes accessible at a, then $\mathcal{U}_a =_{\mathbb{V}} \mathcal{U}'_a$ by definition. This is the uniqueness. The construction of $\mathcal{U}_a$ is similar to the construction of the interpreter $\mathcal{I}_\alpha$. ∎

An interesting corollary of Theorem 19 is the following.

Corollary 20. *$\mathbb{V}$ is equivalent to a submodel of $\mathbb{V}$ with a finite set of symbols for private channel.*

While we are on the subject we point out that the techniques used in the construction of the universal process can be used to answer a question posed in [5]. It asks if the value-passing calculus with parametric definition is strictly stronger than the value-passing calculus with replication. A parametric definition is defined by an equation

$$D(x_1, \ldots, x_k) = T, \tag{8}$$

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 (8). A parametric definition is a parameterized process. The parameters can be values, as in our case, or channels. In the presence of the naming function the former subsumes the latter. The semantics of $D(x_1, \ldots, x_k)$ is defined by the following.

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

We call $D(t_1, \ldots, t_k)$ the instantiation of the parametric definition at $t_1, \ldots, t_k$. The unfolding of a parametric definition admits dynamic binding in the sense that a private channel may be captured by a localization operator. The so called α-conversion is disowned in our setting. 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 $i_1, \ldots, i_k$.

Replication can be defined in terms of parametric definition. Using the encoding of parametric terms defined in [5] and the technique of Section VI, we can prove the following result that provides a positive answer to the question left in [5].

Theorem 21. *$\mathbb{VPC}$ with parametric definition can be interpreted in $\mathbb{VPC}$ with replication.*

A posteori Theorem 21 has to be true if $\mathbb{V}$ is a universal model. Extending $\mathbb{V}$ with parametric definition adds no additional power to $\mathbb{V}$ for the same reason.

Corollary 22. *Parametric definition is admissible in $\mathbb{V}$.*

Theorem 19 lays down the foundation for a theory of definability of $\mathbb{V}$. We now give an outline of the theory. By Corollary 22 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 [28], [33].

1) *Enumeration Theorem*. There is a k+1-ary parametric definition $\mathcal{D}_k(x, x_1, \ldots, x_k)$ such that $\mathcal{D}_k(i, x_1, \ldots, x_k)$ is equal to the i-th k-ary parametric definition for all i.
2) *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 $\mathcal{D}_{s(k,x_1,\ldots,x_m)}(x_{m+1}, \ldots, x_{m+n})$.
3) *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 familiar argument shows that there exists a $\mathbb{V}$-process that outputs at a particular channel the Gödel index of itself. This example brings back one of the starting points of this paper. In a majority of cases we do not care about the explicit definition of a process of certain functionality. All we need to know is the existence of such a process. The theory of definability, which can be developed in the spirit of recursion theory, is concerned with what can be defined by interaction models.

Thesis on Interaction 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 is an avenue of research that calls for further efforts.

We may extend $\mathbb{V}$ to a formalism that admits nondefinable processes. Let C be an m-ary parametric definition and D be an n-ary parametric definition. We say that C is D-definable if there is a $\mathbb{V}$-term T such that $C =_{\mathbb{V}} (x_1 \dots x_m)(q_1 \dots q_n)(T \mid D(q_1, \dots, q_n))$, where $q_1, \dots, q_n$ are private channels that do not appear in D. In other words, C is reducible to D. As another direction for future study one may look for a theory of degrees of unsolvability in the interactive framework.

VIII. Conclusion

Axiom of Completeness ($\forall \mathbb{M}.\ \mathbb{C} \sqsubseteq \mathbb{M}$) was proposed as a foundational postulate in [8]. 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 present paper. Thesis on Interaction prevents us from introducing models that are not physically implementable. What we have done in this paper is to provide evidence for the universal validity of the thesis. The three classical computation models can all be lifted to interaction models in a natural way. The λ-calculus can be promoted in two different ways, resulting in a functional model and an object oriented one. Needless to say the thesis need be further examined.

Being a very powerful model, $\mathbb{V}$ has very few primitives. The prefix operator is necessary to define (sequential) computation. The concurrent composition operator is a let-it-happen extensional combinator that disowns any intensional meaning. The localization operator goes with the composition operator. The conditional operator is a natural programming construct. 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 not to say that $\mathbb{V}$ is not universal enough. What it really 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 minimal π-calculus can be easily extended to account for these operators.

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 $\mathfrak{e}; \sqsubseteq_{\mathbb{N}}$ 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 on 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.

Acknowledgment

The support from the National Science Foundation of China (61472239, ANR 61261130589, 91318301) is acknowledged.

References

[1] H. Barendregt. The Lambda Calculus: Its Syntax and Semantics. North-Holland, 1984.

[2] G. Boudol. Towards a Lambda Calculus for Concurrent and Communicating Systems. *TAPSOFT'89*, Lecture Notes in Comuter Science 351, 149-161, 1989.

[3] A. Church. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58:345-363, 1936.

[4] U. Engberg and M. Nielsen. A Calculus of Commincating Systems with Label Passing, Report DAIMI PB-208, Computer Science Department, University of Aarhus, 1986.

[5] Y. Fu. The Universal Process. To appear in *Logical Methods in Computer Science*, 2017.

[6] Y. Fu. The Value-Passing Calculus. In *Theories of Programming and Formal Methods*, Lecture Notes in Computer Science 8051, 166-195, 2013.

[7] Y. Fu. Nondeterministic structure of computation. *Mathematical Structures in Computer Science*, 25:1295-1338, 2015.

[8] Y. Fu. Theory of Interaction. *Theoretical Computer Science*, 611:1-49, 2016.

[9] Y. Fu and H. Lu. On the Expressiveness of Interaction. *Theoretical Computer Science*, 411:1387-1451, 2010.

[10] J. Gill. Computational Complexity of Probabilistic Turing Machines. *SIAM Journal Computing*, 6:675-695, 1977.

[11] D. Gorla. Towards a unified approach to encodability and separation results for process calculi. In: CONCUR 2008. Lecture Notes in Computer Science 5201, 492-507, 2008.

[12] K. Gödel. Über Formal Unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme. *Monatshefte für Mathematik und Verwandter Systeme I*, 38:173-198, 1931.

[13] M. Hennessy and A. Ingólfsdóttir. A theory of communicating processes with value-passing. Information and Computation 107, 202-236, 1993.

[14] Hennessy, M., Lin, H. Symbolic bisimulations. Theoretical Computer Science 138, 353-369, 1995.

[15] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[16] S. Kleene. General Recursive Functions of Natural Numbers. *Mathematicsche Annalen*, 112:727-742, 1936.

[17] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[18] R. Milner. Functions as Processes. *Mathematical Structures in Computer Science*, 2:119-146, 1992.

[19] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, 100:1-40 (Part I), 41-77 (Part II), 1992.

[20] R. Milner and D. Sangiorgi. Barbed bisimulation. In: *ICALP'92*, Lecture Notes in Computer Science 623. 685-695, 1992.

[21] 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.

[22] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[23] Palamidessi, C. Comparing the expressive power of the synchronous and the asynchronous π-calculus. Mathematical Structures in Computer Science 13:685-719, 2003.

[24] D. Park. Concurrency and Automata on Infinite Sequences. In *Theoretical Computer Science*, Lecture Notes in Computer Science 104, 167-183, 1981.

[25] C. Petri. Communication with Automata. Dissertation, Darmstadt Technical University, 1962.

[26] M. Presburger. Über die Vollständigkeit eines Gewissen Systems der Arithmetik Ganzer Zahlen, in welchem die addition als einzige operation hervortritt. In *Warsaw Mathematics Congress*, 395:92-101, 1929.

[27] L. Priese. On the Concept of Simulation in Asynchronous, Concurrent Systems. *Progress in Cybernatics and Systems Research*, 7:85-92, 1978.

[28] H. Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987.

[29] D. Sangiorgi. Expressing mobility in process algebras: First order and higher order paradigm. Ph.D. thesis, Department of Computer Science, University of Edinburgh, 1992.

[30] D. Sangiorgi. From π-Calculus to Higher Order π-Calculus – and Back. In *Proc. TAPSOFT'93*, Lecture Notes in Computer Science668, 151-166, 1993.

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

[32] E. Santos. Computability by Probabilistic Turing Machines. *Trans. American Mathematical Society*, 159:165-184, 1971.

[33] R. Soare. Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets. Springer-Verlag, Heidelberg, 1987.

[34] B. Thomsen. A calculus of higher order communicating systems. In: Proc. POPL'89. 143-154, 1989.

[35] B. Thomsen. A theory of higher order communicating systems. Information and Computation 116, 38-57, 1995.

[36] A. Turing. On Computable Numbers, with an Application to the Entsheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230-265, 1936.

[37] X. Xu. Distinguishing and relating higher-order and first-order processes by expressiveness. Acta Informatica, 49:445-484, 2012.

[38] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. In *Information Processing'89*, pages 613-618. North-Holland, 1989.

[39] D. Walker. Objects in the π-calculus. Information and Computation, 116:253-271, 1995.