Under consideration for publication in Math. Struct. in Comp. Science

Nondeterministic Structure of Computation

Yuxi Fu[†]

BASICS, Department of Computer Science

Shanghai Jiaotong University, Shanghai 200240

MOE-MS Key Laboratory for Intelligent Computing and Intelligent Systems

Received 8 June 2012; Revised 22 December 2013

Divergence and nondeterminism play a fundamental role in the theory of computation. Their combined effect on computational equality deserves further study. By looking at the issue from the point of view of both computation and interaction, one is led to a canonical equality for nondeterministic computation, revealing its rich algebraic structure. The structure is studied in three ways. Firstly a complete equational system for finite state nondeterministic computation is constructed. The challenge with such a system is to find an equational alternative to the fixpoint induction à la Milner. A negative result, the non-existence of a finite equational system for the canonical equality of nondeterministic computation, is established to support our approach. Secondly infinite state nondeterministic computation is investigated in the light of definability. It is shown that every recursively enumerable set is generated by an unobservable process. Thirdly it is proved that, as far as computation is concerned, the effect produced jointly by divergence and nondeterminism is model independent for a large class of process models. In all the studies C-graphs, which are themselves interesting, are used as abstract representations of the computational objects.

1. Introduction

Our conception of computation is independent of individual mental power since the concept is supported by physical realizability. A Turing machine, or its physical implementation, with an input written on its storage tape, constitutes a computational object. An execution of the computational object, driven by physical laws, either terminates in a final computational object, which is the machine with the result value placed on its storage tape, or does not terminate. In a more abstract model, say the λ-calculus (Barendregt 1984), there is no distinction between machine and datum. Computational objects can take many different shapes and forms. The essence of a computational object is that it can be physically implemented and can be executed by resorting to physical laws. The

[†] fu-yx@cs.sjtu.edu.cn

two kinds of computational objects we have just mentioned are dramatically different. From the point of a user, a terminating computation, or execution, provides a result, whereas a divergent, or nonterminating, computation can never do him/her any good. In the physical world, a divergent computation is bound to go bankrupt, at least theoretically, before it exhausts all the energy of the universe. A model theoretical treatment to computation must distinguish these two kinds of computational objects from the outset.

Modern computing environments feature interactions. What comes with interaction is nondeterminism. In such an environment a computational object may be capable of producing a result, and at the same time it also has the potential to diverge. The choice between the two actions is nondeterministic. This computational object differs from a terminating computational object and a divergent computational object in both the human world and the physical world. In fact the situation is far more complicated. Nondeterminism and divergence can be combined in a very complex way. It is not at all clear if anything can be said about these computational objects.

This paper sets out to study the nondeterministic computational objects. Our objective is to answer the following questions:

— What is the right notion of equality for the nondeterministic computational objects? A reasonable answer to the question is the starting point for subsequent investigations.
— What are the structures of the nondeterministic computational objects? More specifically are there any characterizations of the equivalence classes of these objects?
— What are the nondeterministic computational objects defined in a computation or interaction model? Do they differ from one model to another?

We document in this paper the results we have obtained by looking into these problems. Our contributions are summarized as follows:

— We will present a coherent account of the following interrelated concepts: Church-Turing thesis, bisimulation, computation, interaction, divergence, nondeterminism, computational equality, observational equivalence, Turing completeness. The equality for the nondeterministic computational objects will emerge from the uniform account.
— We will construct a complete equational system for the equality we introduced on finite state computational objects. The system contains a few axiom schemata. We will prove a negative result saying that there does not exist any finite axiom system for the finite state computational objects.
— We will study infinite state computations from the point of view of definability. A general definability result will be established.
— We will show that the nondeterministic structure of computation is model independent for a general class of models.

The rest of the paper is organized as follows. Section 2 discusses equality properties for deterministic computation. Section 3 extends these properties to take into account of interaction. Section 4 characterizes finite state computational objects. Section 5 investigates infinite state computational objects. Section 6 establishes the model independence property of the nondeterministic computational objects. Section 7 points out some further research issues.

2. Church-Turing Thesis and Bisimulation

Our understanding of effective calculability was greatly enhanced in the 1930's by the study of computation models (Kleene 1981). Church's simple and elegant model, the well-known λ-calculus (Church 1936; Cardone and Hindley 2009), turned out to be extremely expressive from the programming point of view. Gödel's recursive functions (Gödel 1931; Davis 1965), known as primitive recursive functions after (Kleene 1936a), evolved to Herbrand-Gödel's general recursive functions (see Gödel's 1934 lecture notes reprinted in (Davis 1965)), although at the time Gödel himself wasn't very sure that general recursion captures all possible effective calculability. The year 1936 was when the foundation of computation theory began to take shape. Kleene (Kleene 1936a) provided a characterization of general recursive functions in terms of primitive recursive functions and the least number operator (the μ-operator). Church (Church 1936) and Kleene (Kleene 1936b) showed that the general recursive functions are in fact the same as the λ-definable functions. Turing, having worked independently on his machine model (Turing 1936; Turing 1937b) and having learnt about the work on λ-definability and general recursion, gave a proof of the equivalence between Turing computability and λ-definability (Turing 1937a). These equivalence results led researchers, in particular Gödel, to believe in the Church-Turing Thesis (called Church Thesis in (Kleene 1952)). Later it was pointed out by Kleene (Kleene 1938) that the thesis also covers the situation of partially defined recursion. Turing remarked in (Turing 1937a) that the λ-calculus is more convenient. What he couldn't say at the time is that, owing to this simplicity, the λ-calculus plays a foundational role among (functional) programming languages (Abramsky 1988). Gödel was reported as saying that Turing machines offer the most convincing formalization of mechanical procedures (Davis 1965). It came as no surprise that the physical feasibility of Turing machines is exploited in the influential von Neumann architecture. After the discoveries of the 1930's, models of computation with different emphases were proposed and proved equivalent to the earlier models of Gödel, Church, Kleene and Turing. Post Systems (Post 1943) and Markov Algorithms (Markov 1960) can be seen as language models, while counter machines (like Minsky Machines (Minsky 1967) and the Unlimited Register Machines of Shepherdson and Sturgis (Shepherdson and Sturgis 1965)) and Random Access Machines (Cook and Reckhow 1973) are members of the class of register machines.

Our understanding of effective calculability can be further enhanced by revealing the hidden depth of the Church-Turing Thesis. The thesis was originally formulated in terms of computable functions. It is normally stated as follows:

CTT-I. The set of functions definable in a model of computation is precisely the set of computable functions.

According to the thesis, the computable functions are precisely those definable by the partial recursive functions (Rogers 1987). The equivalence proofs since the 1930's that support the Church-Turing Thesis reveal a lot more than CTT-I. In all cases there is actually an effective translation from one computation model to another that preserves and reflects computations. To describe the general phenomenon in a *model independent* way, we need to be a bit more formal about computation models and translations between them. We assume that every model is a pair $\langle C_{\mathbb{A}}, \rightarrow_{\mathbb{A}}\rangle$, where $C_{\mathbb{A}}$ is the set of *computational objects*, or *configurations*, and $\rightarrow_{\mathbb{A}}$, whose subscript will be omitted, is the *one step computation* relation. Let $\mathfrak{C}$ be the class of all computation models, ranged over by $\mathbb{A}, \mathbb{B}$. The reflexive and transitive closure of $\rightarrow$ will be denoted by $\rightarrow^*$. Now suppose $\mathcal{T}$ is a translation from $\mathbb{A}$ to $\mathbb{B}$ and $M \rightarrow M'$ is a one step computation in $\mathbb{A}$. The preservation property of $\mathcal{T}$ requires that $\mathcal{T}(M) \rightarrow N_1 \rightarrow N_2 \rightarrow \ldots \rightarrow N_k$, for some computational objects $N_1, \ldots, N_k$ of $\mathbb{B}$, such that the sequence of transitions from $\mathcal{T}(M)$ simulates $M \rightarrow M'$. One difficulty in proving the correctness of $\mathcal{T}$ is in showing that N_k is essentially the same as $\mathcal{T}(M')$. This requires the introduction of an equivalence on the computational objects of $\mathbb{B}$, which would complicate the issue from the outset. An elegant way to bypass this problem is to think of $\mathcal{T}$ as a recursive relation rather than as a recursive function. To make the following account simpler, we abuse notation by identifying $C_{\mathbb{A}}$ with the set of the Gödel indices of the computational objects of $\mathbb{A}$. A translation from $\mathbb{A}$ to $\mathbb{B}$ can then be regarded as a binary relation on the set $\mathbf{N}$ of natural numbers. For every computational object M of $\mathbb{A}$, the interpreter should be able to generate effectively the interpretations of M. Suppose N is one interpretation of M and $N \rightarrow N'$. The correctness of the interpreter also requires that it should be capable of effectively finding a computational object M' in $\mathbb{A}$ such that $M \rightarrow^* M'$ and N' is an interpretation of M'. This is usually called reflection property. These remarks lead to Definition 1 introduced below. Henceforth the set $\mathbf{N}$ is ranged over by i, j, k. Suppose $\mathcal{S}$ is a binary relation. The infix notation $x\mathcal{S}y$ stands for the membership predicate $(x, y) \in \mathcal{S}$, and $\mathcal{S}^{-1}$ denotes the reverse relation $\{(y, x) \mid x\mathcal{S}y\}$. The domain $d(\mathcal{S})$ of $\mathcal{S}$ is $\{x \mid \exists y.x\mathcal{S}y\}$ and the range $r(\mathcal{S})$ of $\mathcal{S}$ is $\{y \mid \exists x.x\mathcal{S}y\}$. The relation $\mathcal{S}$ is *total* if $\forall x.\exists y.x\mathcal{S}y$.

Definition 1. A relation $\mathcal{S} \subseteq \mathbf{N} \times \mathbf{N}$ is *recursive* if the following are valid:

1 Both $d(\mathcal{S})$ and $r(\mathcal{S})$ are recursive.
2 There are (partial) recursive functions $\mathsf{s}^+, \mathsf{s}^- : \mathbf{N} \rightarrow \mathbf{N}$, called *encoding* and *decoding* functions respectively, such that $d(\mathsf{s}^+) = d(\mathcal{S})$ and $d(\mathsf{s}^-) = r(\mathcal{S})$.
3 $\forall x \in d(\mathcal{S}).\mathsf{s}^-(\mathsf{s}^+(x)) = x$.

By definition the composition of two recursive relations is a recursive relation. To appreciate the above definition, notice that the condition (3) of the above definition rules out nonsensical translations like the one translating every divergent Turing machine configuration to $(\lambda x.xx)(\lambda x.xx)$ and every terminating Turing machine configuration to $\lambda x.x$. Studies in complexity theory and programming activities in the real world have led us to believe that the translation overheads, as well as the simulation costs, between computation models are actually polynomially bounded (van Emde Boas 1990; Wegener 2005). In this paper we ignore this *quantitative* aspect of the Church-Turing Thesis.

We are now in a position to define in a model independent manner translations from one computation model to another, formalizing a number of proposals summarized in (van Emde Boas 1990).

Definition 2. Suppose $\mathbb{A}, \mathbb{B}$ are computation models. A relation $\mathcal{T}$ from $C_{\mathbb{A}}$ to $C_{\mathbb{B}}$ is an *effective reduction* of $\mathbb{A}$ to $\mathbb{B}$ if the following statements are valid:

1 $\mathcal{T}$ is total.
2 $\mathcal{T}$ is recursive.
3 If $M\mathcal{T}N$, then $\exists M'.M \rightarrow^* M'\mathcal{T}N'$ whenever $N \rightarrow N'$, and $\exists N'.N \rightarrow^* N'\mathcal{T}^{-1}M'$ whenever $M \rightarrow M'$.
4 If $M\mathcal{T}N$, then there is an infinite sequence of computation starting from M if and only if there is an infinite sequence of computation starting from N.

We write $\mathbb{A} \leq_e \mathbb{B}$ if there is an effective reduction from $\mathbb{A}$ to $\mathbb{B}$.

As we have alluded to in the above, Definition 2 makes little sense without condition 2. Condition (3) is the weak bisimulation property (Park 1981; Milner 1989a) well-known in logic and process theory (Sangiorgi 2009). We will come back to this point later. Conditions (2) and (3) are the fundamental *qualitative* properties of $\mathcal{T}$. Condition (4), usually referred to as the *termination preservation* condition, says that a computational object of $\mathbb{A}$ is undefined if and only if its interpretation in $\mathbb{B}$ is undefined. There are other ways to deal with undefinedness. But the identification of undefinedness with nontermination, or divergence as it is normally called, is the simplest model independent treatment for undefinedness. It is important to realize that if Definition 2 is strengthened by adding the quantitative condition saying that the length of a simulating sequence is polynomial on the length of the computation sequence being simulated, then condition (4) is a corollary of condition (3) if both $\mathbb{A}$ and $\mathbb{B}$ are deterministic. In a quantitative approach terminating computations must be interpreted by terminating computations, and consequently divergent computations must be interpreted by divergent computations. Termination preservation is subordinate to condition (3) in a quantitative framework.

The equivalence proofs that substantiate CTT-I suggest that every one of the well-known (deterministic) computation models can be regarded as an initial model. From the point of view of this paper it is most convenient to take a Counter Machine Model as the initial model $\mathbb{R}$. A CM consists of a finite number of *registers* $r_1, \ldots, r_{k'}$ and a *program*, the latter being a finite sequence of *instructions* $L_1, \ldots, L_k$ with the line numbers $1, 2, \ldots, k$ respectively. An instruction is in one of three forms: $Succ(r_j)$ increments the number in r_j by one before moving to the next instruction; $DecJump(r_j, s)$ decreases the number in r_j by one if it is not zero, or it jumps to the s-th instruction; and *End* stops the execution of the program. Without loss of generality we assume that an instruction of a program is *End* if and only if it is the last instruction of the program. The set $C_{\mathbb{R}}$ of the computational objects of $\mathbb{R}$ is a CM configuration $\langle n_1, \ldots, n_{k'}; i\rangle$, where n_j, for $j \in \{1, \ldots, k'\}$, is the current number stored in the j-th register and i is the location of the current instruction. The one step computation $\langle n_1, \ldots, n_{k'}; i\rangle \rightarrow \langle n'_1, \ldots, n'_{k'}; i'\rangle$ indicates the change of configuration after the execution of the current instruction.

We can now reformulate CTT-I in terms of the effective reduction.

CTT-II. $\forall \mathbb{A} {\in} \mathfrak{C}.\, \mathbb{R} \leq_e \mathbb{A}$.

Conditions (3,4) of Definition 2 are formulated for deterministic computation. This is not really a restriction for CTT-II since $\mathbb{R}$ is deterministic. We shall see in the next section how $\leq_e$ can be strengthened so that it applies to every pair of deterministic/nondeterministic computation models.

3. Church-Turing Thesis and Process Equality

Let's now turn to nondeterministic computations. In a semantic setting, nondeterminism is about state change. Suppose M is a configuration of a nondeterministic Turing machine and that M may in one step evolve nondeterministically to two distinct configurations M', M''. If M' and M'' always lead to final configurations with the same numbers in the output tapes, one may think of these two configurations as semantically equivalent. In this case the nondeterminism is fake. True nondeterminism comes into the picture when M', M'' are not semantically equivalent. This is the situation in which it is possible for M' for instance to reach a final configuration with a number on the output tape whereas M'' can never reach a final configuration with the same output number.

Nondeterminism is not a concept about effectiveness. One can never argue in favour of it from the point of view of effective calculability. Nondeterminism is inevitable if computations are defined in an interactive framework. In the broader picture of modern computing (distributed and mobile computing), everything is about interaction (Milner 1993) and computation is a special form of interaction (Milner 1992; Cai and Fu 2011). Milner pioneered the study of interaction models with his work on the well-known process calculus $\mathbb{CCS}$ (Milner 1989a). A major contribution of the study of process calculi is the theory of observational equivalence. Before explaining the relationship between the observational theory and the Church-Turing Thesis, we need to introduce the $\mathbb{CCS}$ model.

Crucial to the definition of $\mathbb{CCS}$ is the notion of (channel) name. We summarize the relevant terminology concerning names and name variables below.

— The countable set $\mathcal{N}$ of names is ranged over by a, b, c, d, e, f, g, h.
— The set $\overline{\mathcal{N}}$ of co-names is $\{\overline{a} \mid a \in \mathcal{N}\}$.
— The set of actions $\mathcal{A} = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ is ranged over by λ and its decorated versions. The notation τ stands for an internal action. It is different from any name and co-name.
— The set $\mathcal{N}_v$ of name variables is ranged over by u, v, w, x, y, z.
— The union $\mathcal{N} \cup \mathcal{N}_v$ is ranged over by l, m, n, o, p, q.
— The set $\mathcal{L} = \mathcal{A} \cup \mathcal{N}_v \cup \{\overline{x} \mid x \in \mathcal{N}_v\}$ is ranged over by ℓ.

A finite sequence of names $c_1, \ldots, c_k$ for example is sometimes abbreviated to $\widetilde{c}$.

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

$$\begin{array}{rcl} T & := & S \mid T \,|\, T' \mid (c)T \mid D(n_1, \ldots, n_k), \\ S & := & \mathbf{0} \mid \ell.T \mid S + S'. \end{array}$$

In $\ell.T$ we say that ℓ is a prefix of T. The term $\ell_1.T_1 + \ell_2.T_2 + \ldots + \ell_n.T_n$ will be called a Σ-term or a choice term. We will write $\sum_{1 \le i \le n} \ell_i.T_i$ for $\ell_1.T_1 + \ell_2.T_2 + \ldots + \ell_n.T_n$. We have omitted the brackets in $\ell_1.T_1 + \ell_2.T_2 + \ldots + \ell_n.T_n$ since the operator '+' is commutative and associative. The notation $\sum_{i \in I} \ell_i.T_i$ will also be used, where the indexing set I is finite. If the indexing set I is empty, $\sum_{i \in I} \ell_i.T_i$ is understood as $\mathbf{0}$. A trailing $\mathbf{0}$ will often be omitted. The operator '$|$' composes two terms into one, allowing the components to interact. In the localization term $(c)T$ the name c is bound. For simplicity we will abbreviate $(c_1) \ldots (c_k)T$ to $(c_1 \ldots c_k)T$. We avail ourselves of the α-conversion, which says that a local name in a term can be renamed to a fresh name without changing the syntax of the term. The α-conversion is used whenever it is necessary to prevent name capture. The term $D(n_1,\ldots,n_k)$ is the instantiation of a parametric definition $D(x_1,\ldots,x_k)$ at $n_1,\ldots,n_k$. A k-ary parametric definition $D(x_1,\ldots,x_k)$ is given by an equation

$$D(x_1,\ldots,x_k) = T \tag{1}$$

such that the set of the name variables appearing in T is a subset of $\{x_1,\ldots,x_k\}$. The instantiation of $D(x_1,\ldots,x_k)$ at $n_1,\ldots,n_k$ is $T\{n_1/x_1,\ldots,n_k/x_k\}$. We will write A, B, C, D for processes defined by parametric definition. In (1) the parameters $x_1,\ldots,x_k$ are bound. A name variable is free if it is not bound. A $\mathbb{CCS}$ process is a $\mathbb{CCS}$ term that does not contain any free variables. The set of processes will be ranged over by L, M, N, O, P, Q.

The labeled transition semantics of $\mathbb{CCS}$ is generated by the following rules.

Action

$$\frac{}{\lambda.T \xrightarrow{\lambda} T} \qquad \frac{S_1 \xrightarrow{\lambda} S_1'}{S_1 + S_2 \xrightarrow{\lambda} S_1'} \qquad \frac{S_2 \xrightarrow{\lambda} S_2'}{S_1 + S_2 \xrightarrow{\lambda} S_2'}$$

Composition

$$\frac{T_0 \xrightarrow{\lambda} T_0'}{T_0 \,|\, T_1 \xrightarrow{\lambda} T_0' \,|\, T_1} \qquad \frac{T_0 \xrightarrow{a} T_0' \quad T_1 \xrightarrow{\overline{a}} T_1'}{T_0 \,|\, T_1 \xrightarrow{\tau} T_0' \,|\, T_1'}$$

$$\frac{T_1 \xrightarrow{\lambda} T_1'}{T_0 \,|\, T_1 \xrightarrow{\lambda} T_0 \,|\, T_1'} \qquad \frac{T_0 \xrightarrow{\overline{a}} T_0' \quad T_1 \xrightarrow{a} T_1'}{T_0 \,|\, T_1 \xrightarrow{\tau} T_0' \,|\, T_1'}$$

Localization

$$\frac{T \xrightarrow{\lambda} T'}{(c)T \xrightarrow{\lambda} (c)T'} \ c \text{ does not appear in } \lambda.$$

Recursion

$$\frac{T\{n_1/x_1,\ldots,n_k/x_k\} \xrightarrow{\lambda} T'}{D(n_1,\ldots,n_k) \xrightarrow{\lambda} T'} \ D(x_1,\ldots,x_k) = T.$$

In the recursion rule $T\{n_1/x_1,\ldots,n_k/x_k\}$ is the term obtained from T by substituting $n_1,\ldots,n_k$ for $x_1,\ldots,x_k$ simultaneously.

If $T \xrightarrow{a} T_1$, meaning that T can evolve into T_1 by performing the *external* action a, and $T' \xrightarrow{\overline{a}} T_1'$, meaning that T' can turn into T_1' by performing the *external* co-action $\overline{a}$, then T and T' may interact at the channel a, resulting in an *internal* action $T' \,|\, T \xrightarrow{\tau} T_1' \,|\, T_1$. The relation $\xrightarrow{\tau}$ is in fact the one-step computation relation. For example $\tau.a \xrightarrow{\tau} a$ is a deterministic computation step, whereas $\tau.a + \tau.b \xrightarrow{\tau} a$ is a nondeterministic computation step. The reflexive and transitive closure of $\xrightarrow{\tau}$ is denoted by $\Longrightarrow$. The composition $\Longrightarrow\xrightarrow{\lambda}\Longrightarrow$ is abbreviated to $\overset{\lambda}{\Longrightarrow}$.

Among the several variants of $\mathbb{CCS}$ used by Milner (Milner 1989a), the one with the fixpoint operator is relevant to the present work. The grammar of this model is given by

$$\begin{aligned} T &:= X \mid S \mid T \,|\, T' \mid (c)T \mid \mu X.T, \\ S &:= \mathbf{0} \mid \lambda.T \mid S + S'. \end{aligned}$$

In the fixpoint term $\mu X.T$ the free *term variable* X in T gets bound in $\mu X.T$. Notice that in this variant, henceforth denoted by $\mathbb{CCS}^{\mu}$, there is no need for name variables. The semantics of the fixpoint term is defined by the following rule

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

In $\mathbb{CCS}$ the fixpoint term $\mu X.T$ can be simply defined by $D = T\{D/X\}$. So we can use the fixpoint notation in $\mathbb{CCS}$. In (Fu and Lu 2010) it is shown that $\mathbb{CCS}^{\mu}$ is strictly less expressive than $\mathbb{CCS}$. In fact $\mathbb{CCS}$ is Turing complete (Busi, Gabbrielli and Zavattaro 2003; Busi, Gabbrielli and Zavattaro 2004) whereas $\mathbb{CCS}^{\mu}$ is not even Turing complete (Fu and Lu 2010). Milner in his book (Milner 1989a) used another variant, the one with dynamic binding, that is equivalent to $\mathbb{CCS}$ (Giambiagi, Schneider and Valencia2004). We shall use $\mathbb{CCS}^{\mu}$ to characterize the finite state computation, and use $\mathbb{CCS}$ to study the infinite state computation. A more restricted recursion is given by the replication operator. A replication term is of the form $!\lambda.T$, whose semantics is given by the rule

$$\frac{}{!\lambda.T \xrightarrow{\lambda} T \mid !\lambda.T}$$

Clearly $!\lambda.T$ can be defined by $\mu X.\lambda.(T \mid X)$.

Had the development of process theory paralleled that of computation theory, one would have seen the introduction of 'reduction' between process calculi at an early stage. This is not what happened in reality. The truth is that the equivalence relation, a relatively simple concept in computation theory, becomes the most intriguing issue in process theory (Milner 1980). And it has remained one of the major concerns throughout the development of process theory (Milner 1989a; Hennessy 1988; Baeten and Weijland 1990; Sangiorgi and Walker 2001). This is understandable since one would not know how to define a reduction between two process calculi if one did not know how to define an equivalence on one process calculus. After all an equivalence can be seen as a reduction from one process calculus to itself (Fu 2012). For a long time Milner's weak bisimilarity (Milner 1989a) was the main tool to identify the interactive behaviors of processes (Milner 1993). The relation was originally defined in terms of both internal and external actions. Milner and Sangiorgi (Milner and Sangiorgi 1992) pointed out later that the bisimulation property for the external actions is a derivable property. In other words, bisimulation is about computation. To guarantee that a bisimulation equivalence is observational, it should at least be closed under interactive environment and preserve the ability to interact. Hence the next two definitions.

Definition 3. A relation $\mathcal{R}$ on processes is *extensional* if the following statements are valid: $(c)P \;\mathcal{R}\; (c)Q$ if $P\mathcal{R}Q$; and $(P \mid P') \;\mathcal{R}\; (Q \mid Q')$ if $P\mathcal{R}Q$ and $P'\mathcal{R}Q'$.

Definition 4. A process P is *observable*, notation $P\Downarrow$, if $P \Longrightarrow\xrightarrow{\lambda}$ for some $\lambda \neq \tau$. A process P is *unobservable*, notation $P\not\Downarrow$, if $\neg(P\Downarrow)$. A relation $\mathcal{R}$ on processes is *equipollent* if $(P\Downarrow) \Leftrightarrow (Q\Downarrow)$ whenever $P\mathcal{R}Q$.

The equipollence is the barbed condition of Milner and Sangiorgi (Milner and Sangiorgi

1992). We now define Milner's weak bisimilarity (Milner 1989a) in the style of barbed bisimulation.

Definition 5. The *Milner equality* $\approx$ is the largest relation $\mathcal{R}$ that validates the following statements:

1 $\mathcal{R}$ is reflexive.
2 $\mathcal{R}$ is extensional and equipollent.
3 $\mathcal{R}$ is a *weak bisimulation*. In other words the following statements are valid:
— If $M\mathcal{R}N \xrightarrow{\tau} N'$ then $M \Longrightarrow M'\mathcal{R}N'$ for some M'.
— If $N\mathcal{R}^{-1}M \xrightarrow{\tau} M'$ then $N \Longrightarrow N'\mathcal{R}^{-1}M'$ for some N'.

The equivalence $\approx_\downarrow$ is the largest reflexive, extensional, equipollent weak bisimulation that satisfies the termination preservation property:

4 If $M\mathcal{R}N$, then there is an infinite sequence of τ actions starting from M if and only if there is an infinite sequence of τ actions starting from N.

The resemblance between Definition 2 and Definition 5 deserves comment. The totality condition of the former turns into the reflexivity condition of the latter. This is because the former is a relation on two models, while the latter is a relation on one model. Condition (2) of Definition 2, which is an intensional requirement, becomes condition (2) of Definition 5, which is an observational requirement. The analogy brings out the following criticisms to $\approx$:

1 As we have said, weak bisimulation is essentially a property about deterministic computation. It does not really fit in a definition that is supposed to take good care of the nondeterminism caused by interaction.
2 The condition (4) of Definition 2 is completely ignored by $\approx$. This is a deficiency if process calculi are seen to subsume computation models.

One may cast similar doubts on the equivalence $\approx_\downarrow$.

3 The condition (4) of Definition 2 is formulated for the deterministic computations. Does it still make sense in the presence of nondeterminism?

A well known example that showcases the problem of $\approx$ is given by the equality

$$\tau.(a + \tau.b) + c \approx \tau.(a + \tau.b) + \tau.b + c.$$

The one-step nondeterministic computation

$$\tau.(a + \tau.b) + \tau.b + c \xrightarrow{\tau} b \tag{2}$$

is simulated by the two-step nondeterministic computation

$$\tau.(a + \tau.b) + c \xrightarrow{\tau} a + \tau.b \xrightarrow{\tau} b. \tag{3}$$

The intermediate state $a + \tau.b$ is bisimilar to neither $\tau.(a + \tau.b) + c$ nor b. At state $a + \tau.b$ the environment may well intervene to disrupt the simulation. It is debatable if (3) can really be seen as a simulation of (2). In computation theory the distinction between deterministic computations and nondeterministic computations is of fundamental importance. The issue raised by this example should be properly addressed.

van Glabbeek and Weijland (van Glabbeek and Weijland 1989) proposed a more discriminating approach to nondeterminism. The branching bisimulations they discovered is based on the following Computation Lemma (Fu and Lu 2010), called Stuttering Lemma in (van Glabbeek and Weijland 1989).

Lemma 1. Let $=$ be either the weak bisimilarity or the absolute equality to be defined later. If $P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_n = P_1$, then $P_1 = P_2 = \ldots = P_n$.

The lemma implies that if $P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_k \xrightarrow{\tau} P_{k+1}$ such that P_{k+1} is not equivalent to P_k, then P_{k+1} is not equivalent to P_i for any $i < k$. The philosophy of the branching bisimulation of van Glabbeek and Weijland (van Glabbeek and Weijland 1989) is that a state-change internal action (nondeterministic computation step) is so dramatic that it has to be *bi*simulated in a one-to-one fashion, and that a state-preserving internal action (deterministic computation step) has so little consequence that it can be completely ignored. In the following definition we leave out the adjective 'branching'.

Definition 6. $\mathcal{R}$ is a *bisimulation* if the following statements are valid:

— If $P\mathcal{R}Q \xrightarrow{\tau} Q'$ then one of the following statements is valid:
  - $P \Longrightarrow P'$ for some P' such that $P'\mathcal{R}Q$ and $P'\mathcal{R}Q'$.
  - $P \Longrightarrow P''\mathcal{R}Q$ for some P'' such that $\exists P'.P'' \xrightarrow{\tau} P'\mathcal{R}Q'$.

— If $Q\mathcal{R}^{-1}P \xrightarrow{\tau} P'$ then one of the following statements is valid:
  - $Q \Longrightarrow Q'$ for some Q' such that $Q'\mathcal{R}^{-1}P$ and $Q'\mathcal{R}^{-1}P'$.
  - $Q \Longrightarrow Q''\mathcal{R}^{-1}P$ for some Q'' such that $\exists Q'.Q'' \xrightarrow{\tau} Q'\mathcal{R}^{-1}P'$.

Let's now turn to the second criticism. The equivalence $\approx$ does not differentiate

$$\Omega \stackrel{\text{def}}{=} \mu X.\tau.X \tag{4}$$

from $\mathbf{0}$. One way to strengthen the weak bisimulation is to impose the termination preservation property (Walker 1990; Aceto and Hennessy 1992), as we have done to $\approx_{\downarrow}$. While this condition distinguishes between Ω and $\mathbf{0}$, it identifies the following processes.

$$D_1 = \mu X.(\tau.(X \mid d) + \tau.X + \tau), \tag{5}$$
$$D_2 = \mu X.(\tau.(X \mid d) + \tau). \tag{6}$$

The equivalence $\approx_{\downarrow}$ allows an infinite sequence of deterministic computations

$$D_1 \xrightarrow{\tau} D_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} D_1 \xrightarrow{\tau} \ldots$$

to be simulated by an infinite sequence of nondeterministic computations

$$D_2 \xrightarrow{\tau} D_2 \mid d \xrightarrow{\tau} \ldots \xrightarrow{\tau} D_2 \mid d \mid d \xrightarrow{\tau} \ldots,$$

which cannot be justified since the former produces nothing, whereas the latter keep offering the environment the possibility to interact at channel d.

It is not difficult to see how to modify the termination preservation condition to take nondeterminism into account. Suppose $P_0 \approx Q_0$ and

$$P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_i \xrightarrow{\tau} \ldots \tag{7}$$

is an infinite computation sequence. If (7) contains an infinite number of nondeterministic computation steps, the bisimulation property guarantees that (7) is bisimulated by an infinite computation. However bisimulation fails to achieve that if (7) contains only a finite number of nondeterministic computation steps. The additional requirement is captured by the condition introduced next.

Definition 7. A relation $\mathcal{R}$ is *codivergent* if the following statements are valid:

— If $P_0\mathcal{R}Q_0 \xrightarrow{\tau} Q_1 \ldots \xrightarrow{\tau} Q_n \xrightarrow{\tau} \ldots$, then $\exists P_1.\exists j > 0.P_0 \overset{\tau}{\Longrightarrow} P_1\mathcal{R}Q_j$.
— If $P_0\mathcal{R}^{-1}Q_0 \xrightarrow{\tau} Q_1 \ldots \xrightarrow{\tau} Q_n \xrightarrow{\tau} \ldots$, then $\exists P_1.\exists j > 0.P_0 \overset{\tau}{\Longrightarrow} P_1\mathcal{R}^{-1}Q_j$.

The property described in Definition 7 was introduced by Priese (Priese 1978). It has been rediscovered in different contexts (van Glabbeek, Luttik and Trčka 2009; Fu and Lu 2010).

At last we reach the model independent definition of process equality. We assume that all process models have both the concurrent composition operator and the restriction operator.

Definition 8. The *absolute equality* $=_{\mathbb{M}}$ of a process model $\mathbb{M}$ is the largest binary relation on $\mathbb{M}$ processes that validates the following statements:

1 The relation is reflexive.
2 The relation is extensional and equipollent.
3 The relation is a codivergent bisimulation.

The definition of the absolute equality is uniform for all models. We shall often omit the subscript in $=_{\mathbb{M}}$. For more motivation for absolute equality the reader may consult (Fu 2012). Let's see an interesting equality about unobservable processes.

Lemma 2. $P \,|\, \Omega = \Omega$ for every unobservable process P.

Proof. Let $\mathcal{R}$ be $\{(P_0, P_1) \mid \forall i{\in}\{0,1\}.P_i \equiv \Omega \vee (P \,|\, \Omega \Longrightarrow P_i \text{ and } P \text{ is unobservable})\}$ and let $\mathcal{R}^\circ$ be the least relation satisfying the following: (i) $\mathcal{R} \subseteq \mathcal{R}^\circ$; (ii) $\mathcal{R}^\circ$ is reflexive; (iii) if $P\mathcal{R}^\circ Q$ then $(P \,|\, O)\mathcal{R}^\circ(Q \,|\, O)$ and $(O \,|\, P)\mathcal{R}^\circ(O \,|\, Q)$ for every process O; (iv) if $P\mathcal{R}^\circ Q$ then $(c)P\mathcal{R}^\circ(c)Q$ for every name c. The relation $\mathcal{R}^\circ$ is a reflexive, extensional, equipollent, codivergent bisimulation. □

We can now formally define the terminology we have informally used so far.

— $T \to T'$ if $T \xrightarrow{\tau} T' = T$. We say that $T \to T'$ is a *deterministic computation* step since T and T' must have the same input/output capacities.
— $T \xrightarrow{\iota} T'$ if $T \xrightarrow{\tau} T' \neq T$. We say that $T \xrightarrow{\iota} T'$ is a *nondeterministic computation* step for the reason that either T and T' have different input/output behaviors or they have different divergent behaviors.

In the presence of the bisimulation condition, the codivergence condition is equivalent to the termination preserving condition for deterministic computations. So the two conditions are equivalent when we deal with deterministic computation models.

Let's come back to the examples (5) and (6). One has $D_1 \approx_{\downarrow} D_2$. However $D_1 \neq D_2$

since D_1 can perform an infinite sequence of deterministic computation steps whereas D_2 cannot. It is the codivergence condition that tells them apart.

After Definition 8 we can now define effective translations between computation models that admit nondeterminism.

Definition 9. A relation $\mathcal{R}$ from a model $\mathbb{A}$ to a model $\mathbb{B}$ is an *effective subbisimilarity* if the following statements are valid:

1 $\mathcal{R}$ is total.

2 $\mathcal{R}$ is recursive.

3 $\mathcal{R}$ is a codivergent bisimulation.

We write $\mathbb{A} \sqsubseteq_e \mathbb{B}$ if there is an effective subbisimilarity from $\mathbb{A}$ to $\mathbb{B}$.

The terminology "subbisimilarity" is introduced for the fact that, if $\mathbb{A} \sqsubseteq_e \mathbb{B}$ then there is a bisimulation from $\mathbb{A}$ to $\mathbb{B}$ that reveals a submodel relationship between $\mathbb{A}$ and $\mathbb{B}$.

The relation $\sqsubseteq_e$ applies to a more general situation than $\leq_e$. Obviously $\mathbb{R} \sqsubseteq_e \mathbb{A}$ implies $\mathbb{R} \leq_e \mathbb{A}$. So CTT-II can be strengthened in terms of the more informative relation $\sqsubseteq_e$.

CTT-III. $\forall \mathbb{A} \in \mathfrak{C}. \, \mathbb{R} \sqsubseteq_e \mathbb{A}$.

CTT-III applies to both computation models and interaction models. We say that a model $\mathbb{A}$ is *Turing complete* if $\mathbb{R} \sqsubseteq_e \mathbb{A}$. The transitivity of $\sqsubseteq_e$ allows one to conclude that $\mathbb{B}$ is Turing complete after showing $\mathbb{R} \sqsubseteq_e \mathbb{A} \sqsubseteq_e \mathbb{B}$.

The following particular instance of CTT-III is relevant to the present work. Apart from the argument for the recursiveness the proof is from (Busi, Gabbrielli and Zavattaro 2003).

Proposition 1. $\mathbb{R} \sqsubseteq_e \mathbb{CCS}$.

Proof. The j-th register r_j with the initial value 0 is interpreted by Z_j defined by

$$\begin{aligned} Z_j &= zero_j.Z_j + inc_j.(c)(S_j(c) \,|\, c.Z_j), && (8) \\ S_j(x) &= dec_j.\overline{x} + inc_j.(c)(S_j(c) \,|\, c.S_j(x)). && (9) \end{aligned}$$

The value of the register can be increased and decreased by interacting at channel inc_j and dec_j respectively. The standard form of the interpretation of r_j with value i is the process $(c_{i-1} \ldots c_0)(S_j(c_{i-1}) \,|\, c_{i-1}.S_j(c_{i-2}) \,|\, \ldots \,|\, c_2.S_j(c_1) \,|\, c_1.S_j(c_0) \,|\, c_0.Z_j)$. Suppose $L_1, L_2, \ldots, L_k$ is the program of a CM. For each $i \in \{1, \ldots, k-1\}$ the instruction L_i is interpreted by

$$I_i = \begin{cases} \overline{inc_j}.I_{i+1} & \text{if } L_i = Succ(r_j), \\ \overline{dec_j}.I_{i+1} + \overline{zero_j}.I_s & \text{if } L_i = DecJump(r_j, s). \end{cases} \qquad (10)$$

The last instruction, which is *End*, is simply interpreted as $\mathbf{0}$. To achieve recursiveness we use a fixed countable set of names $\{c_0, c_1, c_2, \ldots\}$. We also need the following commutative and associative rewriting system:

$$\begin{aligned} \mathbf{0} \,|\, T &\mapsto T, \\ (c_i)(T \,|\, T') &\mapsto T \,|\, (c_i)T', && \text{if } c_i \text{ does not appear in } T; \\ (c_i)T &\mapsto T, && \text{if } c_i \text{ does not appear in } T; \\ (c_i)(\overline{c_i} \,|\, c_i.T \,|\, T') &\mapsto (c_i)(T \,|\, T'), && \text{if } c_i \text{ does not appear in } T'. \end{aligned}$$

The reflexive and transitive closure of $\mapsto$ is denoted by $\mapsto^*$. Suppose CM is a CM. The *canonical interpretation* of CM is the tuple

$$\langle N_1, \ldots, N_{k'}, I_j, D(L_1), \ldots, D(L_k)\rangle,$$

where $L_1, \ldots, L_k$ is the program of CM, $D(L_1), \ldots, D(L_k)$ are the parametric definitions interpreting the instructions, I_j is the interpretation of the current instruction, and $N_1, \ldots, N_{k'}$ are the normal forms interpreting the values of the registers of CM. Let $\mathcal{R}$ be the set of the pairs of the form $\langle \mathsf{CM}, \langle N'_1, \ldots, N'_{k'}, I_j, D(L_1), \ldots, D(L_k)\rangle\rangle$ such that $N'_1 \mapsto^* N_1$, $\ldots$, $N'_{k'} \mapsto^* N_{k'}$, where $\langle N_1, \ldots, N_{k'}, I_j, D(L_1), \ldots, D(L_k)\rangle$ is the canonical interpretation of CM. Busi, Gabbrielli and Zavattaro's proof can be strengthened to show that $\mathcal{R}$ is total, codivergent and bisimilar. The recursiveness is due to the following facts:

1 It is decidable if a $\mathbb{CCS}$ process is the interpretation of a CM configuration.
2 Given a $\mathbb{CCS}$ interpretation of a CM configuration, it is algorithmically feasible to recover the CM configuration.

Both (1) and (2) make use of the rewriting system. □

Just as CTT-I allows one to use a recursive function without worrying about how it is defined, CTT-III asserts that in a Turing complete model a process with certain behavior must exist and we may refer to it without explicitly defining it. Let's illustrate this point by an example. Suppose $\mathsf{f}(x)$ is a unary computable function and A is a $\mathbb{CCS}$ process A. Then there exists a parametric definition, notation $\mathsf{C}^{\mathsf{f}}(x, u, y, v, z, w).A$, such that $F \overset{\text{def}}{=} \mathsf{C}^{\mathsf{f}}(in, end, out, stop, dec, zero).A$ behaves as follows: If $\mathsf{f}(k)$ is undefined, then

$$F \underbrace{\rightarrow^* \xrightarrow{in} \ldots \rightarrow^* \xrightarrow{in}}_{k} \xrightarrow{end} = \Omega;$$

if $\mathsf{f}(k)$ is defined, then $F \underbrace{\rightarrow^* \xrightarrow{in} \ldots \rightarrow^* \xrightarrow{in}}_{k} \xrightarrow{end} \underbrace{\rightarrow^* \xrightarrow{out} \ldots \rightarrow^* \xrightarrow{out}}_{\mathsf{f}(k)} \xrightarrow{stop} = A \,|\, [\![k]\!]^{zero}_{dec}$, where $[\![k]\!]^{zero}_{dec} \equiv \underbrace{\overline{dec}. \cdots .\overline{dec}}_{k}.\overline{zero}$. For those who have not built up strong confidence in the interactive version of Turing completeness, we point out that formally the parametric definition $\mathsf{C}^{\mathsf{f}}(x, u, y, v, z, w).A$ is given by

$$(ab)(ak_0)(ak_1)(zr)(ic)(dc)(ic_1)(R \,|\, Z_0 \,|\, [\![\mathsf{f}]\!]^{zr,ic,dc}_{a} \,|\, a.\overline{b}.\overline{ak_1}.A \,|\, b.Cp(dc, zr, y, v) \,|\, R_1),$$

where

$$\begin{array}{rcl}
R & = & u.\overline{ak_0} + x.(c)(S(c) \,|\, c.R), \\
S(z) & = & u.\overline{ic}.\overline{ic_1}.\overline{z}.\overline{u} + x.(c)(S(c) \,|\, c.S(z)), \\
Z_0 & = & ak_0.\overline{zr} + ic.(c)(S_0(c) \,|\, c.Z_0), \\
S_0(x) & = & ak_0.\overline{dc}.\overline{x}.\overline{ak_0} + ic.(c)(S_0(c) \,|\, c.S_0(z))), \\
R_1 & = & ak_1.\overline{w} + ic_1.(c)(S_1(c) \,|\, c.R_1), \\
S_1(x) & = & ak_1.\overline{z}.\overline{x}.\overline{ak_1} + ic_1.(c)(S_1(c) \,|\, c.S_1(z))), \\
C_p(dc, zr, y, v) & = & \overline{zr}.\overline{v} + \overline{dc}.(c)(S_c(c) \,|\, c.R), \\
S_c(z) & = & \overline{zr}.\overline{y}.\overline{z}.zr + \overline{dc}.(c)(S_c(c) \,|\, c.S_c(z)),
\end{array}$$

and $[\![\mathsf{f}]\!]_a^{zr,ic,dc}$ is the process that calculates f with the input number stored in the register with access channels zr, ic, dc. The process indicates the termination of the calculation by performing the action $\overline{a}$, which can be achieved by interpreting the *End* instruction by $\overline{a}$. In summary the process F inputs the number k at channels in, end, and then calculates $\mathsf{f}(k)$; if $\mathsf{f}(k)$ is undefined, then it loops forever; otherwise it outputs the number $\mathsf{f}(k)$ at channels $out, stop$ and copy the number k at channel $dec, zero$ before it starts A. The number k is kept because later calculation may well need it. It is easily seen how to generalize from $\mathsf{C}^{\mathsf{f}}(x,u,y,v,z,w).A$ to $\mathsf{C}^{\mathsf{f}}(\widetilde{x},\widetilde{u},y,v,\widetilde{z},\widetilde{w}).A$ for a k-ary computable function.

We are now in a position to define reductions between process calculi. Suppose $\mathcal{R}$ is an effective reduction from $\mathbb{M}$ to $\mathbb{N}$ and that $P\mathcal{R}P'$ and $Q\mathcal{R}Q'$. If P and Q are codivergent bisimilar, then P' and Q' are codivergent bisimilar by the definition of $\mathcal{R}$. In other words equal computational objects are translated to equal computational objects by $\mathcal{R}$. We need to promote this soundness condition from the level of computation to the level of interaction.

Definition 10. A relation from the set of $\mathbb{M}$-processes to the set of $\mathbb{N}$-processes is *sound* if $P\mathcal{R}P'$, $Q\mathcal{R}Q'$ and $P =_{\mathbb{M}} Q$ implies $P' =_{\mathbb{N}} Q'$.

The next definition should be compared to Definition 2 and Definition 8.

Definition 11. A relation from the set of $\mathbb{M}$-processes to the set of $\mathbb{N}$-processes is a *subbisimilarity* if it validates the following statements:

1 It is total and sound.
2 It is extensional and equipollent.
3 It is a codivergent bisimulation.

We write $\mathbb{M} \sqsubseteq \mathbb{N}$ if there is a subbisimilarity from $\mathbb{M}$ to $\mathbb{N}$.

Intuitively $\mathbb{M} \sqsubseteq \mathbb{N}$ means that $\mathbb{N}$ is at least as expressive as $\mathbb{M}$. In this paper there are only a few simple references to $\sqsubseteq$. So we shall not elaborate on the relation. See (Fu 2012) for a full exposition of the expressiveness relation.

We now fix the class $\mathfrak{M}$ of interaction models we are concerned with in this paper. The running theme of this paper is that an interaction model is a computation model if the interactive capacity of the model is ignored. Thus $\mathfrak{M} \subseteq \mathfrak{C}$. When thinking of a process model $\mathbb{M}$ as a computational model the following assumptions on $\mathbb{M}$ are general enough to cover all models of our interest.

— There is an effective bijection between $\mathbf{N}$ and the set of the $\mathbb{M}$-processes. In other words the $\mathbb{M}$-processes are Gödel enumerable.
— All computations in $\mathbb{M}$ are finite branching.
— There is an algorithm that, given an $\mathbb{M}$-process P, calculates all the one-step computations of the form $P \xrightarrow{\tau} P'$.

These effective conditions have been used to study structural operational semantics, giving rise to effective operational semantics (Vaandrager 1993). Variants of these conditions are proposed in a number of papers (Baeten, Bergstra and Klop 1987; de Simone 1985; Darondeau 1990; Bloom, Istrail and Meyer 1995).

Our interaction models are expressive enough so that (i) they are Turing complete and (ii) they are at least as expressive as $\mathbb{CCS}$. By Proposition 1, (i) is subsumed by (ii). Hence the next definition.

Definition 12. An interaction model $\mathbb{M}$ rendering true the above three statements is called a *Turing-Milner model* if $\mathbb{CCS} \sqsubseteq \mathbb{M}$.

In a Turing-Milner model the execution tree of an unobservable process can be algorithmically generated. The finite branching versions of the well-known complete process calculi are Turing-Milner models. These include the π-calculus (Milner, Parrow and Walker 1992; Sangiorgi and Walker 2001) and the value-passing calculus (Hennessy and Ingólfsdóttir 1993; Hennessy and Lin 1995; Fu 2013). Proving that a model is not a Turing-Milner model is generally a tricky issue. We know however that $\mathbb{CCS}^{\mu}$ is not a Turing-Milner model since it is not even complete (Fu 2012).

From now on $\mathfrak{M}$ denotes the class of Turing-Milner models. For a model $\mathbb{M}$ in $\mathfrak{M}$, we write $P \in \mathbb{M}$ to indicate that P is an $\mathbb{M}$-process.

What we have done so far is a condensed account of the journey starting from the original formulation of the Church-Turing Thesis, via Milner and Park's notion of weak bisimulation and van Glabbeek and Weijland's discovery of the branching bisimulation, reaching to a technical formalization of the thesis. Our journey was led by the fact that it is in the theory of interaction that the full picture of the nondeterministic computations can be unveiled. It is against this background that Definition 8 emerges as a canonical equality for both interactive objects and computational objects. The purpose of this paper is to start investigating the rich structures of nondeterministic computations revealed by this equality.

4. Finite State Computation

In this section we shall use a variant of $\mathbb{CCS}^{\mu}$ that admits only τ actions. The *finite state terms* are generated from the following grammar:

$$\begin{array}{rcl} T & := & X \mid S \mid \Delta(T) \mid \mu X.T, \\ S & := & \mathbf{0} \mid \tau.T \mid S + S. \end{array}$$

The Δ operator is introduced by Hennessy and Milner (Hennessy 1981), where it is called a delay operator and is denoted by δ. It plays an important role in the algebraic theory of SCCS (Milner 1983). The term $\Delta(T)$ either behaves as T or evolves to itself. A *finite state computational object* is a finite state term that does not contain any free term variables. The semantics of the Δ-operator is defined by the following two rules:

$$\frac{}{\Delta(T) \xrightarrow{\tau} \Delta(T)} \qquad \frac{T \xrightarrow{\tau} T'}{\Delta(T) \xrightarrow{\tau} T'}$$

We call the transition $\Delta(T) \xrightarrow{\tau} \Delta(T)$ a self-loop. We write $T \rightsquigarrow T'$ if $T \rightarrow T'$, that is $T \xrightarrow{\tau} T' = T$, and the transition $T \xrightarrow{\tau} T'$ is not caused by a self-loop. If there does not exist any T' such that $T \rightarrow T'$ then we write $T \nrightarrow$. The notation $T \not\rightsquigarrow$ is defined similarly.

The simplest terminating computational object is of course **0**, and the simplest divergent computational object is Ω defined in (4). All the other finite state computational objects have potential both to terminate and to diverge. The execution of $\Delta(\tau)$ can be described by the following labeled tree:

$$\begin{tikzcd}
\bullet \arrow[r, "\iota"] \arrow[d] & \circ \\
\circ \arrow[r, "\iota"] \arrow[d] & \circ \\
\circ \arrow[r, "\iota"] & \circ \\
\vdots &
\end{tikzcd}$$

where the node indicated by '$\bullet$' is the root of the tree. The object $\tau + \tau.\Omega$ is an internal choice between **0** and Ω. Its execution tree is

$$\begin{tikzcd}
\cdots & \circ & \circ \arrow[l] & \circ \arrow[l] & \bullet \arrow[l, "\iota"'] \arrow[r, "\iota"] & \circ
\end{tikzcd}$$

The execution tree of $\Delta(\tau + \tau.\Omega)$ is

$$\begin{tikzcd}
\cdots & \circ & \circ \arrow[l] & \circ \arrow[l] & \bullet \arrow[l, "\iota"'] \arrow[r, "\iota"] \arrow[d] & \circ \\
\cdots & \circ & \circ \arrow[l] & \circ \arrow[l] & \circ \arrow[l, "\iota"'] \arrow[r, "\iota"] \arrow[d] & \circ \\
\cdots & \circ & \circ \arrow[l] & \circ \arrow[l] & \circ \arrow[l, "\iota"'] \arrow[r, "\iota"] & \circ \\
 & & & & \vdots &
\end{tikzcd}$$

Using the codivergence condition it is easy to see that the above five computational objects are pairwise unequal.

The finite state computational objects would not be very interesting if there are only finitely many of them. We now construct an infinite sequence of finite state computational objects that are pairwise unequal. The first three in the sequence are defined as follows:

$$\begin{aligned}
\Upsilon_0 &= \mathbf{0}, \\
\Upsilon_1 &= \Delta(\tau.\mathbf{0}), \\
\Upsilon_2 &= \tau.\mathbf{0} + \tau.\Omega.
\end{aligned}$$

Starting from Υ_3 the sequence is defined recursively by (11) and (12), where $i > 0$:

$$\Upsilon_{2i+1} = \Delta(\tau.\mathbf{0} + \tau.\Upsilon_{2i}), \tag{11}$$

$$\Upsilon_{2i+2} = \tau.\mathbf{0} + \tau.\Omega + \tau.\Upsilon_{2i+1}. \tag{12}$$

The next theorem says that the processes $\Upsilon_0, \Upsilon_1, \Upsilon_2, \ldots$ are pairwise unequal.

Theorem 1. $\forall i > 0.\forall j < i.\ \Upsilon_j \neq \Upsilon_i$.

Proof. The natural induction is as follows:

— It is clear that $\Upsilon_0, \Upsilon_1, \Upsilon_2$ are pairwise unequal and none of them is equal to any Υ_k for every $k \geq 3$.

— Suppose $k \leq 2i$. By the induction hypothesis the self loop $\Upsilon_{2i+1} \rightarrow \Upsilon_{2i+1}$ cannot be bisimulated by Υ_k if k is even. If k is odd $\Upsilon_{2i+1} \xrightarrow{\tau} \Upsilon_{2i}$ cannot be bisimulated by Υ_k according to the induction hypothesis. So the process Υ_{2i+1} is not equal to any Υ_k with $k \leq 2i$.

— Suppose $k \leq 2i+1$. The action $\Upsilon_{2i+2} \xrightarrow{\iota} \Omega$ cannot be matched up by any action of Υ_k whenever k is odd. If k is even, $\Upsilon_{2i+2} \xrightarrow{\tau} \Upsilon_{2i+1}$ cannot be bisimulated by Υ_k according to the induction hypothesis. So the process Υ_{2i+2} is not equal to any Υ_k with $k \leq 2i+1$.

We are done. □

4.1. *Algebraic Properties*

Since a computational object never interacts, the absolute equality over the computational objects is simply the largest codivergent bisimulation on these objects. We will also use the so called strong bisimilarity (Milner 1980; Park 1981).

Definition 13. The *Milner-Park equality* $\sim$ is the largest relation $\mathcal{R}$ that validates the following *strong bisimulation* property:

— If $P\mathcal{R}Q \xrightarrow{\tau} Q'$, then $P \xrightarrow{\tau} P'\mathcal{R}Q'$ for some P'.

— If $Q\mathcal{R}^{-1}P \xrightarrow{\tau} P'$, then $Q \xrightarrow{\tau} Q'\mathcal{R}^{-1}P'$ for some Q'.

Obviously $\sim \subsetneq = \subsetneq \approx$.

An equality on processes can be extended to an equality on terms in the standard manner. For instance $T = T'$ if $T\{P_1/X_1, \ldots, P_n/X_n\} = T'\{P_1/X_1, \ldots, P_n/X_n\}$ for all processes $P_1, \ldots, P_n$.

Let's now take a look at the algebraic properties of the fixpoint terms modulo the equalities $\sim$, $=$, $\approx$. The following equality is the defining property of the fixpoint $\mu X.T$:

$$\mu X.T \sim T\{\mu X.T/X\}. \tag{13}$$

The term $T\{\mu X.T/X\}$ is said to be the unfolding of $\mu X.T$. The following beautiful axioms of Bloom and Ésik (Bloom and Esik 1994) are also valid with respect to the Milner-Park equality.

$$\mu X.T\{T'/X\} \quad \sim \quad T\{\mu X.T'\{T/X\}/X\}, \tag{14}$$

$$\mu X.\mu Y.T \quad \sim \quad \mu Y.T\{Y/X\}. \tag{15}$$

Thinking of $\mu X.T\{T'/X\}$ as a tree, it is an infinite alternation of the part defined by T and the part defined by T', which explains (14). It should be clear that (13) is a special case of (14). If we admit the unguarded choice terms of the form $T + T'$, we have the following equalities:

$$\mu X.(X{+}T) \quad \sim \quad \mu X.T, \tag{16}$$

$$\mu X.(\tau.X{+}T) \quad \approx \quad \mu X.\tau.T, \tag{17}$$

$$\mu X.(\tau.(X{+}T){+}T') \quad = \quad \mu X.(\tau.X{+}T{+}T'). \tag{18}$$

The equalities (16), (17) and (18) are the famous axioms of Milner (Milner 1989b) for the finite states. They help to remove the unguarded occurrences of a bound variable so that the following *fixpoint induction* rule can be applied.

$$\frac{T' = T\{T'/X\}}{T' = \mu X.T} \quad X \text{ is guarded in } T. \tag{19}$$

By 'X is guarded in T' we mean that every occurrence of X in T is guarded by at least one non-τ prefix.

The equalities (13) through (18) have all been used as axioms in equational systems for the finite state behaviours. If we intend to construct an equational proof system for the absolute equality $=$ on the finite state computational objects, none of (16), (17) and (18) would be useful. This is because (16) and (18) make essential use of the unguarded choice operator and (17) is not even valid for the termination preservation bisimilarity. Needless to say the fixpoint induction rule (19) is also irrelevant in the present context since we have ignored all non-τ actions.

Let's now turn to the algebraic properties of divergence. Milner's approach to obtaining complete axiom systems for finite state behaviours heavily relies on the fixpoint induction rule. If we try to apply it to tackle the divergent problem of the finite state processes, we would face the following dilemma:

— All divergence is essentially caused by the unguarded occurrences of bound variables.
— The side condition of (19) rules out any possibility to reason about divergence using fixpoint induction.

An ingenious solution that bypasses this dilemma is given by Lohrey, D'Argenio and Hermanns (Lohrey, D'Argenio and Hermanns 2002; Lohrey, D'Argenio and Hermanns 2005). Their key observation is that for finite state behaviours all forms of divergence are caused by *τ-loops*, and that all τ-loops can be reduced to *self-loops*. An induced operator that helps rewrite all τ-loops to self-loops is therefore important. Using the unguarded choice operator it can be defined by

$$\Delta(T) \quad \stackrel{\text{def}}{=} \quad \mu X.(\tau.X{+}T), \tag{20}$$

where the variable X does not appear in T. Some useful equalities about the Δ-operator follow immediately from definition (20). For example

$$\Delta(T) \sim \tau.\Delta(T) + T \sim \Delta(T) + T \sim \tau.\Delta(T) + \Delta(T).$$

In this paper we cannot use (20) to define Δ because we do not use the general choice operator. This is why we have introduced Δ as a primitive operator. The equalities studied in (Lohrey, D'Argenio and Hermanns 2002; Lohrey, D'Argenio and Hermanns 2005) are all weaker than the absolute equality. But some of their laws are valid for $=$. Here are two examples.

$$\mu X.(\Delta(X{+}T) + T') \quad = \quad \mu X.\Delta(T{+}T'), \tag{21}$$
$$\mu X.(\tau.(X{+}T) + T') \quad = \quad \mu X.(\tau.\Delta(T{+}T') + T'). \tag{22}$$

Again these laws make use of the unguarded choice.

Let's summarize the issues we must resolve in order to produce an equational system. First of all most laws concerning the μ-operator and the Δ-operator are no longer available. We need to come up with some alternatives. Secondly we need to find an equational replacement for the fixpoint induction. The fixpoint induction allows one to prove the equality between a short term and a very long term. An example is $\mu X.a.X = \mu X.\underbrace{a\ldots a}_{k\text{ times}}.X$ where $k > 0$. It is proved in (Sewell 1994; Sewell 1997) that no pure equational system is strong enough to derive this equality for all $k > 0$. We will give a similar non-existence result in Section 4.3. Our solution is to use axiom schemata. For that purpose one needs to use C-contexts.

Definition 14. A *C-context* is either $[\_]$, or $\tau.C[\_]{+}S$, or $S{+}\tau.C[\_]$, or $\Delta(C[\_])$, or $\mu X.C[\_]$ whenever $C[\_]$ is a C-context.

The next lemma states three useful schemata with the help of C-contexts.

Lemma 3. The following equalities are valid:

1 $\mu X.C[\tau.(\tau.X{+}S_1){+}S_2] = \mu X.C[\tau.X{+}S_1{+}S_2]$;
2 $\mu X.C[\Delta(X)] = \mu X.C[\tau.X]$;
3 $\mu X.C[\Delta(\tau.X{+}S)] = \mu X.C[\tau.X{+}S]$.

Proof. All the three equalities can be established using Computation Lemma. $\square$

4.2. *Axioms for Finite State Computation*

To simplify the description of the computational objects, we have introduced the prefixed version of the binary choice operator and the Δ-operator. The tradeoff is that the absolute equality is no longer a congruence. It is well known that the binary choice operator does not preserve $=$. One has that $\mathbf{0} = \tau$, yet $\mathbf{0} + \tau.\Omega \neq \tau + \tau.\Omega$. The Δ-operator does not preserve $=$ either. The processes τ and $\mathbf{0}$ are equal. But clearly $\Delta(\tau) \neq \Delta(\mathbf{0})$. The largest congruence contained in the absolute equality can be defined in the standard manner.

Definition 15. P and Q are congruent, notation $P \asymp Q$, if the following are valid:

1 If $P \xrightarrow{\tau} P'$ then $Q \xrightarrow{\tau} Q' = P'$ for some Q'.
2 If $Q \xrightarrow{\tau} Q'$ then $P \xrightarrow{\tau} P' = Q'$ for some P'.

Suppose $P \equiv \sum_{i\in I} \tau.T_i$ and $Q \equiv \sum_{j\in J} \tau.T_j$ with $\equiv$ being the syntactic equality, and neither P nor Q contains any occurrences of the μ-operator. If $Q = P \xrightarrow{\iota} P'$ and $Q' \neq P'$ for all Q' such that $Q \xrightarrow{\tau} Q'$, then it is easily seen that $P + \tau.\Upsilon_k \neq Q + \tau.\Upsilon_k$ for some large enough k. This explains why clause (1) of the above definition cannot be replaced by "If $P \xrightarrow{\tau} P'$ then $Q \rightarrow^* \xrightarrow{\tau} Q' = P'$ for some Q'". The proof of the next proposition is standard.

Proposition 2. The equivalence $\asymp$ is the largest congruence contained in $=$.

The proof system AC for finite state computational objects consists of the axioms defined in Fig. 1. We write $AC \vdash T = T'$ to indicate the fact that the equality $T = T'$

S1	$\mathbf{0} + S$	$=$	S
S2	$S_1 + S_2$	$=$	$S_2 + S_1$
S3	$(S_1 + S_2) + S_3$	$=$	$S_1 + (S_2 + S_3)$
S4	$S + S$	$=$	S
F1	$\mu X.T$	$=$	$T\{\mu X.T/X\}$
F2	$\mu X.X$	$=$	$\mathbf{0}$
F3	$\mu X.(\tau.X{+}S)$	$=$	$\Delta(\mu X.S)$
F4	$\mu X.C[\tau.(\tau.X{+}S_1){+}S_2]$	$=$	$\mu X.C[\tau.X{+}S_1{+}S_2]$
F5	$\mu X.C[\Delta(X)]$	$=$	$\mu X.C[\tau.X]$
F6	$\mu X.C[\Delta(\tau.X{+}S)]$	$=$	$\mu X.C[\tau.X{+}S]$
D1	$\Delta(\Delta(T))$	$=$	$\Delta(T)$
D2	$\Delta(S)$	$=$	$S + \tau.\Delta(S)$
T	$\tau.T$	$=$	$\tau.\tau.T$
C	$\tau.(S{+}S')$	$=$	$\tau.(S{+}\tau.(S{+}S'))$

Fig. 1. Axioms for Computation

can be derived by repetitive use of the equivalence laws, the congruence laws and the laws of AC.

The axiom C is related to the B law of van Glabbeek and Weijland (van Glabbeek and Weijland 1989) by restricting to τ-prefix. The T law is a special case of Milner's first tau law. A special case of C is

$$\tau.S \quad = \quad \tau.(S + \tau.S). \tag{23}$$

F2 deals with bound variables that are not prefixed. F3 and F4 can be applied to get rid of a prefixed bound variable. F5 and F6 help to remove a Δ-operator outside a bound variable. These axioms and axiom schemata should be compared to the laws (21) and (22) of Lohrey, D'Argenio and Hermanns (Lohrey, D'Argenio and Hermanns 2002; Lohrey, D'Argenio and Hermanns 2005). For the above system the fixpoint unfolding law F1 can be simplified to

$$\mu X.T = T$$

with the side condition that X does not appear in T. The axiom D1, appeared in Hennessy's work more than 30 years ago (Hennessy 1981), is weaker than the following axiom of Lohrey, D'Argenio and Hermanns:

$$\Delta(\Delta(T) + T') = \tau.(\Delta(T) + T'). \tag{24}$$

The equality (24) fails the codivergence condition.

Proposition 3. If $AC \vdash T = T'$ then $T \asymp T'$.

Proof. F3 is obviously valid if X does not appear in S. If X occurs in S then $\mu X.(\tau.X + S) \asymp \mu X.S$. So clearly F3 also holds in this case. F4, F5 and F6 are valid by Lemma 3. D1 and D2 are actually true of the Milner-Park equality. □

$$\begin{tikzcd}
\circlearrowright & \circ \arrow[l] \arrow[r] & \circ \\
\circlearrowright & \bullet \arrow[l] \arrow[u] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\circlearrowright & \circ \arrow[l] \arrow[r] & \circ \\
\circlearrowright & \underset{\circlearrowright}{\bullet} \arrow[l] \arrow[u] &
\end{tikzcd}$$

Fig. 2. Two D-Graphs

The following equality follows from F3, F1, S1 and S2.

$$\Delta(\mathbf{0}) = \mu X.\tau.X. \tag{25}$$

An instance of F5 is

$$\mu X.\Delta(X) = \mu X.\tau.X. \tag{26}$$

The law D2 is as it were a divergent counterpart of the C law. It implies the equality

$$\Delta(\mathbf{0}) = \tau.\Delta(\mathbf{0}). \tag{27}$$

4.2.1. *D-Normal Form* The finite state computational objects can be visualized as graphs. Conversely finite graphs of a certain type can be coded up by the finite state computational objects. It is trivial to generalize the finite graphs that represent the finite state computational objects to infinite graphs.

Definition 16. A directed graph is a *D-graph* if the following hold:

1 There is a special node called the *root* of the D-graph.
2 There is at most one edge from one node to another node. A *self-loop* is an edge from a node to itself.
3 Every node is reachable from the root.

The two diagrams in Fig. 2 are D-graphs. A root is indicated by a '$\bullet$'. If the root self-loops, a '$\circlearrowright$' is placed right below the '$\bullet$'. If a node is not a root, it is indicated by a '$\circ$'; and if it self-loops, it is simply indicated by a '$\circlearrowright$'.

Definition 17. A D-graph is *definable* in a process model if there is an unobservable process P of that model that is codivergent bisimilar to the root of the D-graph. In this case we say that the D-graph *represents* the process. We write $\mathfrak{d}(P)$ for the D-graph generated by P.

Every finite D-graph is definable by a finite state computational object. We start by introducing a variable for each node of a given finite D-graph. If the D-graph has n nodes, we define a set of n equations of the form:

$$X_0 = T_0,\ \ldots,\ X_{n-1} = T_{n-1}.$$

The first is the head equation corresponding to the root. If the i-th node has no outgoing edges then we have $X_i = \mathbf{0}$, and if the node has a self-loop then we introduce the equation $X_i = \Delta(\mathbf{0})$. If the i-th node has k children $X_{m_1}, \ldots, X_{m_k}$, then the i-th equation is $X_i = \tau.X_{m_1} + \ldots + \tau.X_{m_k}$. If in addition the i-th node has a self-loop, the i-th equation is $X_i = \Delta(\tau.X_{m_1} + \ldots + \tau.X_{m_k})$. Using the standard method (Milner 1984) we can easily construct solutions to the equation system. The solution to X_0 is represented by the D-graph.

We now describe a procedure that converts a finite computational object to one that corresponds to a D-graph. For that purpose we introduce D-normal forms.

Definition 18. A term T is a *D-normal form*, or *D-nf*, if it satisfies the following:

1 T is a variable;
2 If T is a Σ-term, say $\sum_{i\in I}\tau.T_i$, then T_i must be a D-nf for every $i \in I$; in particular $\mathbf{0}$ is a D-nf;
3 If T is a Δ-term then it must be either of the form $\Delta(X)$ or of the form $\Delta(S)$ such that S is a D-nf;
4 If T is a μ-term then it must be of the form $\mu X.S$ such that S is a D-nf and X appears in S;
5 T contains no subterm of the form $\Delta(X)$ for any bound variable X.

The finite state $\mu X.\mu Y.(\tau.X+\tau.Y)$ for example is equal to $\mu X.(\Delta(\mu Y.\tau.X))$ by F3, which is in turn converted to $\mu X.(\Delta(\tau.X))$ by F1, which can be further reduced to $\mu X.\tau.X$ by F6. We conclude that $AC \vdash \mu X.\mu Y.(\tau.X + \tau.Y) = \mu X.\tau.X = \Delta(\mu X.X) = \Delta(\mathbf{0})$ by F2, F3, S1 and S2.

A D-nf is a $\mathbb{CCS}$ process notation for a finite D-graph. The process $\mathbf{0}$ is the trivial one-node D-graph, and Ω is the D-graph with only one node and the self-loop. The Σ-process $\sum_{1\le i\le n}\tau.T_i$ defines the D-graph whose root has n out-going edges pointing to the roots of the D-graphs defining $T_1,\ldots,T_n$ respectively. The Δ-process $\Delta(S)$ corresponds to the D-graph whose root has a self-loop. The μ-process $\mu X.S$ is obtained from the D-graph of $S\{\mathbf{0}/X\}$ by adding the obvious edges pointing to the root.

Lemma 4. For each term T, there is a D-nf T' such that $AC \vdash T = T'$.

Proof. According to the structural definition of D-nf, a subterm of a D-nf is a D-nf and if T, T' are in D-nf then $T\{T'/X\}$ is a D-nf. The proof of the lemma is given by the following structural induction.

— $T \equiv \mathbf{0}$ or $T \equiv X$. There is nothing to prove.
— $T \equiv \tau.T'$. We apply the induction hypothesis to T'.
— $T \equiv \sum_{i\in I}\tau.T_i$. We apply the induction hypothesis to each T_i. Notice that we can apply S1 to remove redundant $\mathbf{0}$ from a Σ-term.
— $T \equiv \Delta(T')$. By the induction hypothesis some D-nf T'' exists such that $AC \vdash T' = T''$. If T'' is a Δ-term, we apply D1 to remove the extra Δ-operator. If T'' is a μ-term, we apply F1 to unfold it.
— $T \equiv \mu X.T'$. By the induction hypothesis, $AC \vdash T' = T''$ for some D-nf T''. If X does not occur in T'', we get rid of the μ-operator by applying F1. If T'' is X, we apply F2 to get $\mathbf{0}$. If T'' is $\Delta(X)$, we apply (26) to get Ω. If T'' is a μ-term, we apply F1 to unfold it. If T'' is of the form $\Delta(S)$, then we apply F6 to remove the Δ-operator. Finally use F5 to remove the Δ-operator in all occurrences of $\Delta(X)$.

We are done. □

4.2.2. *C-Normal Form* Two nodes in a D-graph are said to be equal if they are codivergent in the sense of Definition 7 and bisimilar in the sense of Definition 6. The left

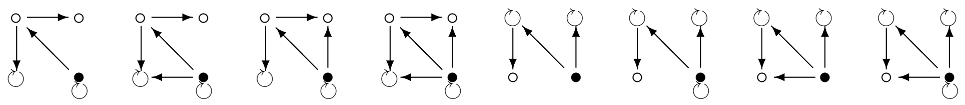
Fig. 3. C-Graphs with Four Nodes

D-graph in Fig. 2 has two pairs of equal nodes; the right D-graph has one pair of equal nodes. The structure of computations is given by the D-graphs containing no equal nodes.

Definition 19. A *C-graph* is a D-graph in which no two nodes are equal.

The eight D-graphs in Fig. 3 are C-graphs with four nodes.

It follows from definition that a C-graph does not contain any loops larger than a self-loop. In particular the root of a C-graph does not have any incoming edges apart from a possible self-loop.

Let $\mathfrak{G}$ be the set of C-graphs, ranged over by $\mathfrak{f}, \mathfrak{g}, \mathfrak{h}, \mathfrak{i}$. The letter $\mathfrak{f}$ will stand for a finite C-graph and $\mathfrak{i}$ an infinite one. For each $k > 0$ let $\mathfrak{F}_k$ be the set of the finite C-graphs with k-nodes. The C-graph representing Υ_k has precisely $k+1$ nodes. So $\mathfrak{F}_k$ is a nonempty finite set for each $k > 0$.

Lemma 5. The roots of two C-graphs are equal if and only if the C-graphs are isomorphic.

Proof. Suppose $\mathfrak{g}_0, \mathfrak{g}_1$ are C-graphs. If they are isomorphic, then one can construct a codivergent bisimulation by paring all the corresponding nodes. Conversely suppose the roots of the graphs are equal. Given any node N in $\mathfrak{g}_0$, a unique path from the root to N exists. Using the bisimulation property one can construct a unique corresponding path in $\mathfrak{g}_1$. The end node of this path corresponds to N. This gives rise to a function from the set of nodes of $\mathfrak{g}_0$ to the set of nodes of $\mathfrak{g}_1$. By induction one easily sees that it is an isomorphic map. □

According to Lemma 5, the C-graph representation of an unobservable process P is unique. Consequently we can talk about equality between C-graphs. We write $\mathfrak{c}(P)$ for this unique C-graph.

We are now going to single out the set of finite state computational objects that correspond to finite C-graphs. The correspondence is not as close as that between the D-nf's and the finite D-graphs. In the process algebraic notation there is no way to remove all the repetitive occurrences of a subterm.

Definition 20. A D-nf T is a *C-normal form*, or *C-nf*, if the following are valid:

1 T does not contain any occurrences of the μ-operator;

2 One of the following statements is true:

(a) $T \equiv X$ or $T \equiv \Delta(X)$ for some term variable X;

(b) $T \equiv \sum_{i\in I} \tau.T_i$ such that, for each $i \in I$, the term T_i is a C-normal form; moreover $\forall j, k \in I.j \neq k \Rightarrow T_j \neq T_k$ and $\forall i \in I.T_i \neq T$;

(c) $T \equiv \Delta(\sum_{i\in I} \tau.T_i)$ such that $\sum_{i\in I} \tau.T_i$ is a C-nf and $\forall i \in I.T_i \neq T$.

Conditions (2a,2b) make sure that $T \not\leadsto$, meaning that T contains no loops other than self-loops. If T_1, T_2 are C-normal forms then $T_1 \asymp T_2$ is almost the same as $T_1 \sim T_2$. This suggests that the C-nf's should play an important role in the completeness proof. The next proposition states a pre-completeness result.

Proposition 4. Suppose T_1, T_2 are C-nf's and $T_1 = T_2$. The following are valid:

1 If both T_1 and T_2 are Σ-terms, then $AC \vdash \tau.T_1 = \tau.T_2$.

2 If both T_1 and T_2 are Δ-terms, then $AC \vdash T_1 = T_2$.

Proof. The proof is carried out by simultaneous structural induction. Since a Δ-term loops while a Σ-term that is also a C-nf does not loop, T_1, T_2 must be both Σ-terms or both Δ-terms.

— $T_1 \equiv \sum_{i\in I} \tau.T_1^i$ and $T_2 \equiv \sum_{j\in J} \tau.T_2^j$ with $I \neq \emptyset \neq J$. The action $T_1 \xrightarrow{\tau} T_1^i$ must be matched by $T_2 \xrightarrow{\tau} T_2^j = T_1^i$ for some T_2^j, according to the definition of C-nf. By the induction hypothesis $AC \vdash \tau.T_1^i = \tau.T_2^j$. It follows that $AC \vdash T_2 + \tau.T_1^i = T_2 + \tau.T_2^j = T_2$. We conclude by induction and symmetry that $AC \vdash T_1 = T_1 + T_2 = T_2$.

— $T_1 \equiv \Delta(\sum_{i\in I} \tau.T_1^i)$ and $T_2 \equiv \Delta(\sum_{j\in J} \tau.T_2^j)$. Since both $\sum_{i\in I} \tau.T_1^i$ and $\sum_{j\in J} \tau.T_2^j$ are C-nf's, it follows easily by a bisimulation argument that $\sum_{i\in I} \tau.T_1^i = \sum_{j\in J} \tau.T_2^j$. So $AC \vdash \sum_{i\in I} \tau.T_1^i = \sum_{j\in J} \tau.T_2^j$ by the induction hypothesis. Hence $AC \vdash T_1 = T_2$.

We are done. □

We will reach our goal if we can prove a normalization result, stating that every term is provably equal to a C-nf. This is indeed the case to a large extent.

Proposition 5. The following statements are valid:

1 For every Σ-term T there is some C-nf T' such that $AC \vdash \tau.T = \tau.T'$.

2 For every Δ-term T there is some C-nf T' such that $AC \vdash T = T'$.

Proof. Observe that we may remove all occurrences of μ-operator in a term by using the F-laws. The following is an algorithm that removes the fixpoint operator in $\mu X.T$.

— If X does not appear in T, apply F1 and exit.

— If $T \equiv X$, apply F2 and exit.

— Suppose X appears in T and $T \not\equiv X$.

- Replace every occurrence of $\Delta(X)$ by $\tau.X$ using F5.
- Now every occurrence of X must appear in some subterm $\tau.X$ of T. We can remove all occurrences of X in the following manner:
  - Apply F4 and F6 to remove all τ-prefixes and Δ's in front of the subterm.
  - If the subterm $\tau.X$ is not guarded by any τ and Δ, apply F3.

  Notice that the transformation does not create any new occurrence of X.

It follows from the observation and the proof of Lemma 4 that we may assume that T is a D-normal form containing no occurrences of the μ-operator. The following argument is by structural induction.

— $T \equiv \mathbf{0}$ or $T \equiv X$. There is nothing to prove.

— $T \equiv \sum_{i\in I} \tau.T_i$. For each $i \in I$ there is by the induction hypothesis a C-nf T'_i such that

$$AC \vdash \tau.T_i = \tau.T'_i. \tag{28}$$

If $T'_i = T'_j$ for some $j \neq i$ then $AC \vdash \tau.T'_i = \tau.T'_j$ according to Proposition 4. So we may remove one of them using S4. Without loss of generality, we assume that for all $i, i' \in I$, $T'_i \neq T'_{i'}$ whenever $i \neq i'$. Now suppose

$$\sum_{i\in I} \tau.T'_i = T'_k \tag{29}$$

for some $k \in I$. If I is a singleton set, then

$$AC \vdash \tau.T = \tau.\tau.T'_k = \tau.T'_k$$

by the T-law. Now suppose the size of I is greater than 1. There are two subcases.

– T'_k is a Σ-term. For every $j \in I \setminus \{k\}$ there is some D-nf T^j_k such that $T'_k \xrightarrow{\tau} T^j_k = T'_j$. So we have

$$AC \vdash \tau.T^j_k = \tau.T'_j \tag{30}$$

by Proposition 4. It follows from (28), (30), S4 and C that

$$\begin{aligned}
AC \vdash \tau.T &= \tau.\left(\tau.T'_k + \sum_{j\in I\setminus\{k\}} \tau.T'_j\right)\\
&= \tau.\left(\tau.\left(T'_k + \sum_{j\in I\setminus\{k\}} \tau.T^j_k\right) + \sum_{j\in I\setminus\{k\}} \tau.T^j_k\right)\\
&= \tau.\left(T'_k + \sum_{j\in I\setminus\{k\}} \tau.T^j_k\right)\\
&= \tau.T'_k.
\end{aligned}$$

– T'_k is $\Delta(S_k)$ for some C-nf S_k. Using a similar argument one can show that

$$AC \vdash \sum_{j\in I\setminus\{k\}} \tau.T'_j = S_k. \tag{31}$$

It follows from (31) that

$$\begin{aligned}
AC \vdash T &= \tau.T'_k + \sum_{j\in I\setminus\{k\}} \tau.T'_j\\
&= \tau.\Delta(S_k) + S_k\\
&= \Delta(S_k)\\
&= T'_k,
\end{aligned}$$

where the second last equality is due to D2. Hence $AC \vdash \tau.T = \tau.T'_k$.

— $T \equiv \Delta(\sum_{i\in I} \tau.T_i)$ where $\sum_{i\in I} \tau.T_i$ is a C-nf. Suppose $\Delta(\sum_{i\in I} \tau.T_i) = T_k$ for some $k \in I$. Since T_k is a C-nf, it must be a Δ-term. Let T_k be $\Delta(S_k)$. Using the above argument we can show that

$$\begin{aligned} AC \vdash S_k &= S_k + \sum_{j\in I\setminus\{k\}} \tau.T_j \\ &= \sum_{j\in I\setminus\{k\}} \tau.T_j. \end{aligned}$$

Therefore

$$\begin{aligned} AC \vdash T &= \Delta\left(\tau.\Delta(S_k) + \sum_{j\in I\setminus\{k\}} \tau.T_j\right) \\ &= \Delta\left(\tau.\Delta(S_k) + S_k\right) \\ &= \Delta(\Delta(S_k)) \\ &= \Delta(S_k), \end{aligned}$$

where the third equality holds by D2 and the fourth equality holds by D1.

This completes the proof. □

4.2.3. *Completeness* We are now in a position to prove the completeness result.

Theorem 2. $T' \asymp T''$ if and only if $AC \vdash T' = T''$.

Proof. Suppose T', T'' are C-nf's and $T' \asymp T''$. The proof is a structural induction.

— If T' is $\mathbf{0}$ then T'' must be $\mathbf{0}$. Similarly if T' is X then T'' must be X.

— If T', T'' are Δ-terms, then $AC \vdash T' = T''$ by Proposition 4 and Proposition 5.

— Suppose $T' \equiv \sum_{i\in I} \tau.T_i \asymp \Delta(S) \equiv T''$. By Proposition 5 and the C law we may assume that $\Delta(S)$ and all T_i's are C-nf's. By definition there is some $k \in I$ such that $T_k = \Delta(S)$. Therefore $T_k \asymp \Delta(S)$ since both T_k and $\Delta(S)$ are C-nf. If $I \setminus \{k\}$ is nonempty, then it is easy to see that $\sum_{i\in I\setminus\{k\}} \tau.T_i \asymp S$. So

$$AC \vdash \sum_{i\in I\setminus\{k\}} \tau.T_i = S$$

by induction. Hence

$$AC \vdash \sum_{i\in I} \tau.T_i = \tau.\Delta(S) + \sum_{i\in I\setminus\{k\}} \tau.T_i = \tau.\Delta(S) + S = \Delta(S),$$

where the last equality is due to D2. Suppose $I \setminus \{k\}$ is empty. Then $\tau.T_k \asymp \Delta(S)$. By codivergence $T_k = \Delta(S)$. Since both T_k and $\Delta(S)$ are C-nf, one must have $T_k \asymp \Delta(S)$. If $S \equiv \mathbf{0}$ then we can use (27) to finish the proof. If S is not $\mathbf{0}$, it must be a proper choice term. By a bisimulation argument and induction one easily sees that $AC \vdash T_k = \tau.\Delta(S) + S = \Delta(S)$.

— If both T', T'' are Σ-terms, then $AC \vdash T' = T''$ follows from Proposition 4, Proposition 5 and the C law.

The proof is complete. □

It can be easily verified that $\mu X.\Delta(T) \asymp \Delta(\mu X.T\{\Delta(X)/X\})$. So Theorem 2 implies $AC \vdash \mu X.\Delta(T) = \Delta(\mu X.T\{\Delta(X)/X\})$. Using this equality we can push all occurrences of Δ up to the root and down to the leaves.

4.3. *Nonaxiomatisability*

One may feel a little unhappy about the C-contexts which appear in the axiomatic system defined in Fig. 1. The fact is, however, that the use of such contexts is unavoidable. The situation reminds one of Sewell's remarkable result (Sewell 1994; Sewell 1997) on the nonexistence of a finite purely equational system for finite state processes. We shall establish a nonexistence result concerning *finite pure axiom systems.* In our setting a pure axiom system is a *recursive* set of equations between the finite state computational objects. The equations in a pure axiom system are pure as it were since schematic axioms like F4, F5 and F6 are banned.

The proof of Sewell's negative result is highly nontrivial. Fortunately we have a simple proof of a nonexistence result in the present case. The intuition is that for every finite pure axiom system there is a large enough number k such that

$$\mu X.(\tau.(\tau.(\ldots\tau.(\tau.X+\tau.\Upsilon_1)+\ldots)+\tau.\Upsilon_{k-1})+\tau.\Upsilon_k) = \Delta(\tau.\Upsilon_1+\ldots+\tau.\Upsilon_k) \quad (32)$$

is not provable in the system.

Theorem 3. There does not exist any sound and complete finite pure axiom system for $\asymp$ on the finite state terms.

Proof. For the purpose of this proof, let's define the size of a term as the maximum number of nested τ prefixes that guard a variable. Given any finite pure axiom system AC_p there is a number k such that the size of the term on the left/right hand side of any axiom of AC_p is less than k. Now suppose

$$\mu X.(\tau.(\tau.(\ldots\tau.(\tau.X+\tau.\Upsilon_1)+\ldots)+\tau.\Upsilon_{k-1})+\tau.\Upsilon_k) = T'. \quad (33)$$

We prove by induction on derivation that T' must be a μ-term whose size is at least k. If (33) is derived by the transitive rule, we apply the induction hypothesis. If (33) is derived by an application of an axiom in AC_p then, since the size of the axiom is less than k, the term T' must be of the form $\mu X.T_c$ for some T_c and we must have

$$AC_p \vdash \tau.(\tau.(\ldots\tau.(\tau.X+\tau.\Upsilon_1)+\ldots)+\tau.\Upsilon_{k-1})+\tau.\Upsilon_k = T_c.$$

The variable X must appear in T_c for otherwise we could instantiate the X on the left hand side by a term with sufficiently many nested τ prefixes such that a computation sequence from the left hand side cannot be matched up by any computational sequences from the right hand side. We claim that every occurrence of X in T_c is prefixed by τ at least k times. If not then

$$T_c \underbrace{\xrightarrow{\tau}\ldots\xrightarrow{\tau}}_{j\ \text{times}} X \quad (34)$$

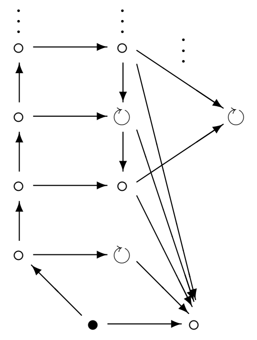
Fig. 4. An Infinite C-Graph

for some $j < k$. Assume that (34) was bisimulated by

$$\tau.(\tau.(\ldots\tau.(\tau.X + \tau.\Upsilon_1) + \ldots) + \tau.\Upsilon_{k-1}) + \tau.\Upsilon_k \underbrace{\xrightarrow{\tau} \ldots \xrightarrow{\tau}}_{j' \text{ times}} T_d$$

for some T_d and some $j' \leq j$ such that

$$T_d = X. \tag{35}$$

It is apparent that (35) is a contradiction. Either T_d does not contain any occurrence of X or every occurrence of X in T_d is prefixed by at least one τ. In either case T_d cannot be equal to X. We conclude that if (33) is derivable from AC_p then T' must be of the form $\mu X.T_c$ such that the size of T_c is at least k. It follows from induction that (32) is not provable in AC_p. □

Theorem 3 adds considerable weight to the system defined in Fig. 1. For further study one could carry out an investigation of the system using the approach of Mendler and Lüttgen (Mendler and Lüttgen2010).

5. Infinite State Computation

We take a look at infinite C-graphs in this section. Our first infinite C-graph is given in Fig. 4. The reader might have noticed that this example is motivated by the sequence $\Upsilon_0, \Upsilon_1, \Upsilon_2, \ldots$. Starting from the root, the computation can travel as high as it is necessary. Once it takes a horizontal move, it reaches a node whose number of descendants is finite. Two distinct nodes on the left vertical chain are unequal since the one nearer to the root can make a move to the right, which cannot be bisimulated by the other. The nodes on the right vertical chain are pairwise distinct because they represent essentially the sequence $\Upsilon_0, \Upsilon_1, \Upsilon_2, \ldots$. A node on the left chain is unequal to a node on the right chain for the reason that the former can engage in an infinite computation composed of nondeterministic steps whereas the latter cannot do that. As this example shows the nodes in an infinite C-graph can be classified into two categories, those that admit an infinite sequence of nondeterministic computations and those that do not.

$$\begin{tikzcd}
\mathfrak{f}_0 & \mathfrak{f}_1 & \mathfrak{f}_2 & \mathfrak{f}_3 & \mathfrak{f}_4 & \cdots \\
\bullet \arrow[u] \arrow[r] & \circ \arrow[u] \arrow[r] & \circ \arrow[u] \arrow[r] & \circ \arrow[u] \arrow[r] & \circ \arrow[u] \arrow[r] & {}
\end{tikzcd}$$

Fig. 5. Construction of a General Infinite C-Graph

Definition 21. A node in a C-graph is *finite* if the number of the nodes it can reach is finite. The *rank* of a finite node N is the size of $\{N' \mid N \stackrel{\tau}{\Longrightarrow} N'\}$. An *infinite* node is a node that is not finite. A finite node is *extremal* if there is an arrow going from an infinite node to the finite node.

The rank of a finite C-graph is the rank of its root. For example the rank of Υ_k is k for each $k \geq 0$.

Suppose $\mathfrak{i}$ is an infinite C-graph. Let $\mathfrak{i}^\omega$ denote the subgraph of $\mathfrak{i}$ consisting of all the infinite nodes of $\mathfrak{i}$. The following lemma follows immediately from definition.

Lemma 6. Suppose $\mathfrak{i}$ is an infinite C-graph. If none of the nodes of $\mathfrak{i}$ has an infinite number of outgoing edges pointing to extremal nodes, then every finite path in $\mathfrak{i}^\omega$ can be extended.

Proof. The assumption of the lemma simply says that $\mathfrak{i}^\omega$ is nonempty. The conclusion of the lemma follows from the fact that every infinite node has at least one child and all of its children in $\mathfrak{i}^\omega$ are infinite. □

Obviously a finite branching infinite C-graph satisfies the property described in the above lemma.

Can an infinite C-graph contain only a finite number of finite nodes? Before answering the question we explain a general construction of an infinite C-graph demonstrated in Fig. 5. Suppose $\mathfrak{f}_0, \mathfrak{f}_1, \mathfrak{f}_2, \ldots$ is an infinite sequence of finite C-graphs such that there is an infinite number of pairwise unequal finite C-graphs, each occurring only finitely often in the sequence. Imagine that the k-th vertical arrow points to the root of $\mathfrak{f}_k$. As it stands, the diagram of Fig. 5 is an infinite D-graph. If we coerce all the equal nodes in the D-graph we get an infinite C-graph. We need to argue that the nodes in the infinite computation $\bullet \to \circ \to \circ \to \ldots$ do not shrink to finite nodes. For that purpose let's name these nodes $N_0, N_1, N_2, N_3, \ldots$. For each $i \geq 0$ the node N_i can reach some root of $\mathfrak{f}_j$ that occurs a finite number of times in $\mathfrak{f}_0, \mathfrak{f}_1, \mathfrak{f}_2, \ldots$. Let k be large enough such that $\mathfrak{f}_j$ no longer appears in $\mathfrak{f}_k, \mathfrak{f}_{k+1}, \mathfrak{f}_{k+2}, \ldots$. Clearly N_i is not equal to N_k. If $\mathfrak{f}_0, \mathfrak{f}_1, \mathfrak{f}_2, \ldots$ are pairwise distinct, then they are pairwise unequal according to Lemma 5. In this case the nodes in $\bullet \to \circ \to \circ \to \ldots$ are pairwise unequal. Once we know that $\bullet \to \circ \to \circ \to \ldots$ does not shrink to a finite sequence, we immediately know that for each $k \geq 0$ the node N_k is not equal to any nodes in any of $\mathfrak{f}_0, \mathfrak{f}_1, \mathfrak{f}_2, \ldots$ since the former admits an infinite nondeterministic computation whereas the latter does not.

Theorem 4. An infinite C-graph has an infinite number of extremal nodes.

Proof. Suppose towards a contradiction there is an infinite C-graph $\mathfrak{i}$ containing a finite number of extremal nodes. Then none of the nodes of $\mathfrak{i}$ has an infinite number of outgoing edges pointing to the extremal nodes. It follows from Lemma 6 that every path of $\mathfrak{i}^\omega$ can be extended to an infinitely long path from the root. Suppose the finite C-graphs rooted by these extremal nodes are $\mathfrak{f}_1, \mathfrak{f}_2, \ldots, \mathfrak{f}_k$. Let's say that $\mathfrak{f}_i$, where $1 \le i \le k$, *belongs* to a node N of $\mathfrak{i}^\omega$ if there is a path $N \xrightarrow{\iota} \ldots \xrightarrow{\iota} N'$ in $\mathfrak{i}^\omega$ such that N' has an outgoing edge pointing to the root of $\mathfrak{f}_i$. Now define the subgraphs $\mathfrak{i}^\omega_0, \mathfrak{i}^\omega_1, \ldots, \mathfrak{i}^\omega_k$ of $\mathfrak{i}^\omega$ successively by the following induction:

— Let $\mathfrak{i}^\omega_0$ be $\mathfrak{i}^\omega$;

— For $i \in \{1, \ldots, k\}$, if there is some node N_i in $\mathfrak{i}^\omega_{i-1}$ to which $\mathfrak{f}_i$ does not belong, then let $\mathfrak{i}^\omega_i$ be the subgraph of $\mathfrak{i}^\omega_{i-1}$ rooted by N_i; otherwise let $\mathfrak{i}^\omega_i$ be $\mathfrak{i}^\omega_{i-1}$.

By assumption $\mathfrak{i}^\omega_k$ is infinite. It is easy to see that the relation consisting of all the pairs of the nodes of $\mathfrak{i}^\omega_k$ is a codivergent bisimulation. In other words all the nodes of $\mathfrak{i}^\omega_k$ are in fact equal. This is a contradiction. □

5.1. *Definability*

Let's now turn to the issue of definability. Is the behavior of the infinite C-graph in Fig. 4 definable in $\mathbb{CCS}$? The answer is positive. Consider the process *Centipeda* defined as follows:

$$Centipeda \;=\; (inc)(dec)(odd)(even)(Cp \mid Cnt \mid even.E \mid odd.O),$$

where

$$\begin{aligned}
Cp &= \tau.\Upsilon_0 + \tau.(\tau.\Upsilon_1 + \tau.(\overline{even} \mid !even.\overline{inc}.\overline{odd} \mid !odd.\overline{inc}.\overline{even})), \\
Cnt &= inc.(d)(A(d) \mid d), && (36)\\
A(x) &= dec.\overline{x} + inc.(d)(A(d) \mid d.A(x)), && (37)\\
O &= \mu X.(\tau.X + \tau + \overline{dec}.E), && (38)\\
E &= \tau + \tau.\Omega + \overline{dec}.O. && (39)
\end{aligned}$$

The component Cp is the main process of *Centipeda*. It admits an infinite computation. At each point of the infinite computation it may branch into a computational object equal to Υ_k for some $k > 0$. The part $\overline{odd} \mid !odd.\overline{inc}.\overline{even} \mid !even.\overline{inc}.\overline{odd}$ either increments the value of the one-time counter Cnt, or fires O or E depending on the parity of the current value of Cnt. Once O (or E) has started, the counter can only decrements. The local name dec is used to control the depth of the recursive calls of E and O. The recursive definition given by (38) and (39) makes clear the relationship to (11) and (12).

Conceivably a C-graph of a computational nature is definable. In what follows we substantiate this intuition.

Definition 22. The *number set* $\mathfrak{n}(\mathfrak{g})$ of a C-graph $\mathfrak{g}$ is defined as follows: A natural number k is in $\mathfrak{n}(\mathfrak{g})$ if and only if there is an extremal node of $\mathfrak{g}$ that is of rank k. A set of natural number is *generated* by an unobservable $\mathbb{CCS}$ process P, denoted by $\mathfrak{n}(P)$, if it is the set $\mathfrak{n}(\mathfrak{c}(P))$.

The number sets of C-graphs do not really capture any nondeterminism. Nonetheless they can be used to indicate the richness of the nondeterministic structures of C-graphs.

$$\begin{tikzcd}
\mathfrak{f}_0 & \mathfrak{f}_1 & \mathfrak{f}_2 & \mathfrak{f}_3 & \mathfrak{f}_4 & \cdots \\
\bullet \arrow[u] \arrow[r, shift left] & \circ \arrow[u] \arrow[l, shift left] \arrow[r, shift left] & \circ \arrow[u] \arrow[l, shift left] \arrow[r, shift left] & \circ \arrow[u] \arrow[l, shift left] \arrow[r, shift left] & \circ \arrow[u] \arrow[l, shift left] \arrow[r, shift left] & {} \arrow[l, shift left]
\end{tikzcd}$$

Fig. 6. Construction of a General Infinite Branching C-Graph

Theorem 5. Every recursively enumerable set containing zero is generated by an unobservable $\mathbb{CCS}$ process represented by a finitely branching C-graph.

Proof. Suppose R is a recursively enumerable set containing zero and χ is the partial characteristic function of R. By definition $x \in R$ if and only if $\chi(x)$ is defined and is equal to 1. Let G be the process given by the following parametric definitions.

$$\begin{aligned}
G &= (in)(end)(dec)(odd)(even)(f)(Reo \,|\, F \,|\, even.E \,|\, odd.O), \\
Reo &= (e)(o)(\overline{in}.\overline{in}.\overline{e} \,|\, !e.\overline{in}.\overline{o} \,|\, !o.\overline{in}.\overline{e} \,|\, e.\overline{end}.f.\overline{even} \,|\, o.\overline{end}.f.\overline{odd}), \\
F &= (out)(stop)(zero)\mathsf{C}^{\chi}(in, end, out, stop, dec, zero).\overline{f},
\end{aligned}$$

where E and O are defined in (38) and (39) respectively, and F is defined in terms of the general construction described on page 13. The role of *Reo* is to randomly generate a number greater than one and record the parity of the number. The process F simulates the calculation of the function χ. When the simulation terminates, it invokes either O or E. Finally let Gr be either $\tau.\Omega + \tau.(\tau.\Upsilon_1 + \tau.G)$ or $\tau.\Omega + \tau.G$, depending on whether $1 \in R$ or not. Clearly the process Gr generates the set R. $\square$

5.2. *Infinite Branching C-Graph*

If we coerce all the equal nodes of a finite branching infinite D-graph, we may get an infinite branching C-graph. That is why a D-graph is not required to be finite branching. A typical construction of a finite branching D-graph equivalent to an infinite branching C-graph is described in Fig. 6. By the Computation Lemma all the nodes in $\bullet \rightleftarrows \circ \leftrightarrows \circ \leftrightarrows \ldots$ are equal.

Theorem 6. There are infinite branching C-graphs definable in $\mathbb{CCS}$.

Proof. Suppose $\{k_0, k_1, k_2, \ldots\}$ is a recursively enumerable set. Then there is a unary *total* recursive function f such that the range of f is $\{k_0, k_1, k_2, \ldots\}$ and that for each $j \geq 0$ there are infinite number of i's such that $\mathsf{f}(i) = k_j$. Let F be $(a)(b)\mathsf{C}^{\mathsf{f}}(in, end, out, zr, a, b)$, where $\mathsf{C}^{\mathsf{f}}(x, u, y, v, z, w)$ is the parametric definition introduced on page 13. Let *Qng* be $(c)(\overline{c} \,|\, !c.\overline{in}.\overline{c} \,|\, c.\overline{end})$. The process *Qng* is a quasi number generator since it may never stop. Let Cq be the process

$$\overline{zr}.\Upsilon_0 + \overline{out}.(\overline{zr}.\Upsilon_1 + \overline{out}.(e)(o)(\overline{e} \,|\, !e.\overline{dec}.(\overline{zr}.\overline{even} + \overline{out}.\overline{o}) \,|\, !o.\overline{dec}.(\overline{zr}.\overline{odd} + \overline{out}.\overline{o}))).$$

Finally the process Fr is defined by

$$(odd)(even)(dec)(out)(zr)(in)(end)(Qng \,|\, F \,|\, Cq \,|\, even.E \,|\, odd.O),$$

where O and E are defined in (38) and (39) respectively. The process Fr does not reach any finite node before F performs any action. After F has done an action, the system turns into a state equal to Υ_{k_i} for some k_i. Clearly Fr generates the set $\{k_0, k_1, k_2, \ldots\}$. Let Tr be the infinite path caused by Qng starting from the root of the D-graph of Fr. It follows from the assumption on f that for every $i \geq 0$ and every node in Tr there is a path from the node to the root of Υ_{k_i} consisting solely of deterministic computation steps. Using the proof technique of Theorem 4, one can easily prove that all the nodes in Tr are equal. This says that the root of $\mathfrak{c}(Fr)$ points to the root of every graph in $\Upsilon_{k_0}, \Upsilon_{k_1}, \Upsilon_{k_2}, \ldots$. We conclude that the root of $\mathfrak{c}(Fr)$ is infinite branching. □

The proof of the above theorem also implies the following result.

Theorem 7. Each recursively enumerable set is generated by an unobservable $\mathbb{CCS}$ process.

6. Church-Turing Thesis and Nondeterminism

Do the unobservable processes defined in one model differ from those definable in another model? More specifically is there a C-graph definable in the π-calculus (Milner, Parrow and Walker 1992) that is not definable in $\mathbb{CCS}$? It is tempting to start one's investigation by trying to answer the second question. It turns out however that the technique allowing one to answer the specific question also help answer the general question to one's satisfaction. The following theorem tells us that the unobservable processes definable in one Turing-Milner model are definable in another Turing-Milner model.

Theorem 8. $\forall \mathbb{M}\in\mathfrak{M}.\forall P\in\mathbb{M}.(P\not\Downarrow \Rightarrow \exists Q\in\mathbb{CCS}.(Q\not\Downarrow \wedge Q = P))$.

Proof. Suppose $\mathbb{M} \in \mathfrak{M}$. The relation $\mathbb{CCS} \sqsubseteq \mathbb{M}$ immediately implies that for each unobservable $\mathbb{CCS}$ process P there is some unobservable $\mathbb{M}$-process Q that is codivergent bisimilar to P. We have the computable functions and the corresponding $\mathbb{CCS}$ processes described below:

— Let n be the unary computable function that, given the Gödel index of an unobservable $\mathbb{M}$-process P, outputs the number of the one-step computations of the form $P \xrightarrow{\tau} P'$. Let $\mathsf{C}^{\mathsf{n}}(x, u, b, e, c, f).A$ be obtained from the general construction defined on page 13.

— Let p be the binary function that, given a number i and the Gödel index of an unobservable $\mathbb{M}$-process P, produces the Gödel index of the $\mathbb{M}$-process P' such that $P \xrightarrow{\tau} P'$ is the i-th one-step computation of P. Let the $\mathbb{CCS}$ process

$$\mathsf{C}^{\mathsf{p}}(h_1, h_2, c, f, a, d, b^+, e^+, c, f).A$$

be obtained from the general construction defined on page 13, where the pair h_1, h_2 and the pair c, f are used to access the input numbers, a, d are used to deliver the result, and b^+, e^+, c, f keep the input numbers.

— Let $\mathsf{C}^0(b^+, e^+)$ be the $\mathbb{CCS}$ process whose sole function is to output the number '0' at channels b^+, e^+.

— Let $\mathsf{C}^{+1}(b^+, e^+, b, e, h_1, h_2, b, e).\overline{g_2}$ be the $\mathbb{CCS}$ process that increments the number at channels b^+, e^+ by one if the number stored at b^+, e^+ is less than the number stored at channels b, e, and then puts the result at channels h_1, h_2 before firing A. The number at channels b, e is kept unchanged.

Now the parametric definition $\mathsf{S}(x,u)$ is given by

$$\begin{aligned}\mathsf{S}(x,u) \quad = \quad & (becf)\mathsf{C}^{\mathsf{n}}(x,u,b,e,c,f).(g)(g_1g_2)(h_1h_2)(b^+e^+)(\overline{g}\,|\,\overline{g_1}\,|\,\mathsf{C}^0(b^+,e^+) \\ & |\,!g_1.\mathsf{C}^{+1}(b^+,e^+,b,e,h_1,h_2,b,e).\overline{g_2} \\ & |\,!g_2.(ad)\mathsf{C}^{\mathsf{p}}(h_1,h_2,c,f,a,d,b^+,e^+,c,f).\overline{g_1}.g.\mathsf{S}(a,d)).\end{aligned}$$

Suppose k is the Gödel index of an $\mathbb{M}$-process P. Let $[\![k]\!]_{dec}^{zero}$ be the $\mathbb{CCS}$ encoding of the number k. The process $S(dec, zero)$ in

$$(dec)(zero)([\![k]\!]_{dec}^{zero}\,|\,S(dec,zero)) \tag{40}$$

reads the number k at channels $dec, zero$, and then calculates the number i of P' such that $P \xrightarrow{\tau} P'$ is a distinct transition. If $i = 0$ then the process halts. If $i > 0$ then the process executes the while-command defined by the two replication processes. In the j-th loop of the execution, where $0 < j \leq i$, the Gödel index k_j of the j-th child of P is calculated and stored at the local names a, d. By the end of the execution the following situation occurs:

$$(g)\left(\overline{g}\,|\,(ad)([\![k_1]\!]_a^d\,|\,g.S(a,d))\,|\,\ldots\,|\,(ad)([\![k_i]\!]_a^d\,|\,g.S(a,d))\right). \tag{41}$$

The computation from the process in (40) to the process in (41) is mainly arithmetical. The two processes are equal. Clearly the $\mathbb{CCS}$ process in (41) has i transitions, simulating all the one-step computations of the $\mathbb{M}$-process P. □

Theorem 8 should be interpreted as saying that, from the perspective of the Church-Turing Thesis, the nondeterministic structure of computation is model independent. This result tells us a number of things. Firstly if we want to study nondeterminism of computation, we may choose any Turing-Milner model without losing any generality. Secondly a Turing-Milner model differs from another Turing-Milner model in that they have different sets of external actions. A model is more expressive than another if the external actions of the former are more expressive than those of the latter. Finally the approach to define the universal equality $=$ and the universal expressiveness relation $\sqsubseteq$ in terms of internal actions (computations) is justified. Theorem 8 can be seen as a formal justification of Definition 8 and Definition 11.

Our careful choice of the equality has paid off. The nondeterministic structures of computations revealed by the absolute equality are rich, and at the same time remain invariant in a general class of models. Such a theory is impossible had we used the weak bisimilarity or termination preserving weak bisimilarity. Weak bisimilarity identifies all computational objects. The termination preserving weak bisimilarity tells apart the three computational objects $\mathbf{0}$, Ω and $\tau.\mathbf{0}+\tau.\Omega$, but identifies any other computational object to one of the three.

Before ending this section we mention that investigations similar to those in Section 5 and this section have been carried out for effective operational semantics (de Simone 1984; de Simone 1985; Vaandrager 1993). Further studies are necessary to see if there is any connection between their work and the present work.

7. Remark

Our understanding of the nondeterminism for both finite-state and infinite-state computations has been improved. There is an algorithm to check if a finite D-graph is a C-graph, based on which one can design a brute force algorithm to enumerate all the elements of $\mathfrak{F}_k$ for every $k > 0$. However we do not know any closed formula to calculate the size of $\mathfrak{F}_k$ at the moment. Although this is a problem belonging to enumerative combinatorics, trying to solve the problem will definitely improve our understanding of the finite state computations. The problem of checking the equivalence of two unobservable $\mathbb{CCS}$ processes is likely beyond the arithmetical hierarchy (Srba 2004a). Checking the equality between an unobservable $\mathbb{CCS}$ process and a finite C-graph poses a much less hard challenge (Srba 2004b). A related problem is the regularity problem, which asks if an unobservable $\mathbb{CCS}$ process is equal to some finite C-graph. Further study is necessary to clear up these equivalence checking issues. A problem that deserves further investigation is a complete characterization of the C-graphs definable in Turing-Milner models. In the light of Theorem 8, it is worth the effort to work out the answer to the question.

Acknowledgments

We thank Chaodong He for raising our interest in the unobservable processes. We also thank the members of BASICS for their interest. We owe the two anonymous referees a great deal. Their constructive criticisms and suggestions have helped improve the quality of the paper considerably. The support from the National Natural Science Foundation of China (60873034, 61033002, 91318301) is gratefully acknowledged.

References

Abramsky, S. (1988) The Lazy Lambda Calculus. In D. Turner, editor, *Declarative Programming*, Addison-Wesley, 65–116.

Aceto, L. and Hennessy, M. (1992) Termination, Deadlock, and Divergence. *Journal of ACM*, 39:147–187.

Baeten, J., Bergstra, J. and Klop, J. (1987) On the Consistency of Koomen's Fair Abstraction Rule. *Theoretical Computer Science*, 51:129–176.

Baeten, J. and Weijland, W. (1990) *Process Algebra*, Cambridge Tracts in Theoretical Computer Science 18. CUP.

Barendregt, H. (1994) *The Lambda Calculus: Its Syntax and Semantics.* North-Holland.

Bloom, S. and Ésik, Z. (1994) *Iteration Algebras of Finite State Process Behaviors.*

Bloom, B., Istrail, S. and Meyer, A. (1995) Bisimulation Can't be Traced. *Journal of ACM*, 42:232–268.

Busi, N., Gabbrielli, M. and Zavattaro, G. (2003) Replication vs Rcursive Definitions in Channel Based Calculi. In *Proc. ICALP'03*, Lecture Notes in Computer Science 2719, 133–144.

Busi, N., Gabbrielli, M. and Zavattaro, G. (2004) Comparing Recursion, Replication and Iteration in Process Calculi. In *Proc. ICALP'04*, Lecture Notes in Computer Science 3142, 307–319.

Cai, X. and Fu, Y. (2011) The λ-Calculus in the π-Calculus. *Mathematical Structure in Computer Science*, 21:943–996.

Cardone, F. and Hindley, J. (2009) Lambda Calculus and Combinators in 20th Century. In D. Gabbay and J. Woods, editors, *Handbook of the History of Logic, Volume 5: Logic from Russel to Church*, Elsevier, 723–817.

Church, A. (1936) An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58:345–363.

Cook, S. and Reckhow, R. (1973) Time Bounded Random Access Machines. *Journal of Computer and System Science*, 7:354–375.

Darondeau, P. (1990) Concurrency and Computability. In *Semantics of Systems of Concurrent Processes, Proceedings LITP Spring School on Theoretical Computer Science*, Lecture Notes in Computer Science 469, 223–238.

Davis, M. (1965) *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions.* Raven Press.

van Emde Boas, P. (1990) Machine Models and Simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Algorithm and Complexity, volume A*, Elservier, 65–116.

Fu, Y. and Lu, H. (2010) On the Expressiveness of Interaction. *Theoretical Computer Science*, 411:1387–1451.

Fu, Y. (2012) Theory of Interaction.

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

Giambiagi, P., Schneider, G. and Valencia, F. (2004) On the Expressiveness of Infinite Behavior and Name Scoping in Process Calculi. In *FOSSACS 2004*, Lecture Notes in Computer Science 2987, 226–240.

van Glabbeek, R, Luttik, B. and Trčka, N. (2009) Branching Bisimilarity with Explicit Divergence. *Fundamenta Informaticae*, 93:371–392.

van Glabbeek, R. and Weijland, W. (1989) Branching Time and Abstraction in Bisimulation Semantics. In *Information Processing'89*, North-Holland, 613–618.

Gödel, K. Über Formal Unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme. *Monatshefte für Mathematik und Verwandter Systeme I*, 38:173–198.

Hennessy, M. (1981) A Term Model for Synchronous Processes. *Computation and Control*, 51:58–75.

Hennessy, M. (1988) *An Algebraic Theory of Processes.* MIT Press, Cambridge, MA.

Hennessy, M. and Ingólfsdóttir, A. (1993) A Theory of Communicating Processes with Value-Passing. *Information and Computation*, 107:202–236.

Hennessy, M. and Lin, H. (1995) Symbolic Bisimulations. *Theoretical Computer Science*, 138:353–369.

Kleene, S. (1936) General Recursive Functions of Natural Numbers. *Mathematicsche Annalen*, 112:727–742.

Kleene, S. (1936) λ-Definablity and Recursiveness. *Duke Mathematical Journal*, 2:340–353.

Kleene, S. (1938) On Notation for Ordinal Numbers. *Journal of Symbolic Logic*, 3:150–155.

Kleene, S. (1952) *Introduction to Metamathematics.* Van Nostrand.

Kleene, S. (1981) Origin of Recursive Function Theory. *Annals of the History of Computing*, 3:52–67.

Lohrey, M., D'Argenio, P. and Hermanns, H. (2002) Axiomatising Divergence. In *Proc. ICALP 2002*, Lecture Notes in Computer Science 2380, 585–596.

Lohrey, M., D'Argenio, P. and Hermanns, H. (2005) Axiomatising Divergence. *Information and Computatio*, 203:115–144.

Markov, A. (1960) The Theory of Algorithms. *American Mathematical Society Translations, series 2*, 15:1–14.

Mendler, M. and Lüttgen, G. (2010) Is Observational Congruence on μ-Expressions Axiomatisable in Equational Horn Logic? *Informaiton and Computation*, 208:634–651.

Milner, R. (1980) A Calculus of Communicating Systems. Lecture Notes in Computer Science, 92.

Milner, R. (1983) Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310.

Milner, R. (1984) A Complete Inference System for a Class of Regular Behaviours. *Journal of Computer and System Science*, 28:439–466.

Milner, R. (1989) *Communication and Concurrency.* Prentice Hall.

Milner, R. (1989) A Complete Axiomatization System for Observational Congruence of Finite State Behaviours. *Information and Computation*, 81:227–247.

Milner, R. (1992) Functions as Processes. *Mathematical Structures in Computer Science*, 2:119–146.

Milner, R. (1993) Elements of Interaction. *Communication of ACM*, 36:78–89.

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

Milner, R and Sangiorgi, D. (1992) Barbed Bisimulation. In *Proc. ICALP'92*, Lecture Notes in Computer Science 623, 685–695.

Minsky, M. (1967) *Computation: Finite and Infinite Machines.* Prentice-Hall.

Park, D. (1981) Concurrency and Automata on Infinite Sequences. In *Theoretical Computer Science*, Lecture Notes in Computer Science, 104:167–183.

Post, E. (1943) Formal Reduction of the General Combinatorial Decision Problem. *American Journal of Mathematics*, 65:197–215.

Priese, L. (1978) On the Concept of Simulation in Asynchronous, Concurrent Systems. *Progress in Cybernatics and Systems Research*, 7:85–92.

Rogers, H. (1987) *Theory of Recursive Functions and Effective Computability.* MIT Press.

Sangiorgi, D. (2009) On the Origin of Bisimulation and Coinduction. *Transactions on Programming Languages and Systems*, 31(4).

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

Sewell, P. (1994) Bisimulation is not Finitely (First Order) Equationally Axiomatisable. In *Proc. LICS'94*, IEEE, 62–70.

Sewell, P. Nonaxiomatisability of Equivalence Over Finite State Processes. *Annals of Pure and Applied Logic*, 90:163–191.

Shepherdson, J. and Sturgis, H. (1965) Computability and Recursive Functions. *Journal of Symbolic Logic*, 32:1–63.

de Simone, R. (1984) On Meije and SCCS: Infinite Sum Operators vs. Non-Guarded Definitions. *Theoretical Computer Science*, 30:133–138.

de Simone, R. (1985) Higher-Level Synchronising Devices in Meije-SCCS. *Theoretical Computer Science*, 37:245–267.

Srba, J. (2004) Completeness Results for Undecidable Bisimilarity Problems. *Electronic Notes in Theoretical Computer Science*, 98:5–19.

Srba, J. (2004) Roadmap of Infinite Results. In *Formal Models and Semantics, II.* World Scientific Publishing Co..

Turing, A. (1936) On Computable Numbers, with an Application to the Entsheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265.

Turing, A. (1937) Computability and λ-Definability. *Journal of Symbolic Logic*, 2:153–163.

Turing, A. (1937) On Computable Numbers, with an Application to the Entsheidungsproblem: A Correction. *Proceedings of the London Mathematical Society*, 43:544–546.

Vaandrager, F. (1993) Expressiveness Results for Process Algebras. In *Proceedings REX Workshop on Semantics: Foundations and Applications*, Lecture Notes in Computer Science 666, 609–638.

Walker, D. (1990) Bisimulation and Divergence. *Information and Computation*, 85:202–241.

Wegener, I. (2005) *Complexity Theory*. Springer-Verlag.