Math. Struct. in Comp. Science (2015), *vol.* 25, *pp.* 1295–1338. © Cambridge University Press 2014
doi:10.1017/S0960129514000012 First published online 10 November 2014

Non-deterministic structures of computation

YUXI FU†

BASICS, Department of Computer Science, and
MOE-MS Key Laboratory for Intelligent Computing and Intelligent Systems,
Shanghai Jiaotong University,
Shanghai 200240, China
Email: fu-yx@cs.sjtu.edu.cn

Received 8 June 2012; revised 22 December 2013

Divergence and non-determinism play a fundamental role in the theory of computation, and their combined effect on computational equality deserves further study. By looking at the issue from the point of view of both computation and interaction, we are led to a canonical equality for non-deterministic computation, revealing its rich algebraic structure. We study this structure in three ways. First, we construct a complete equational system for finite-state non-deterministic computation. The challenge with such a system is to find an equational alternative to fixpoint induction *à la* Milner. We establish a negative result in the form of the non-existence of a finite equational system for the canonical equality of non-deterministic computation to support our approach. We then investigate infinite-state non-deterministic computation in the light of definability and show that every recursively enumerable set is generated by an unobservable process. Finally, we prove that, as far as computation is concerned, the effect produced jointly by divergence and non-determinism is model independent for a large class of process models.
We use C-graphs, which are interesting in their own right, as abstract representations of the computational objects throughout the paper.

1. Introduction

Our conception of computation is independent of individual consciousness since the concept is supported by physical realisability. 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 data.

Computational objects can take many different shapes and forms, but their essence is that they can be physically implemented and executed by resort to physical laws. The two kinds of computational objects mentioned above are dramatically different. From the users' point of view, a terminating computation, or execution, provides a result, whereas a divergent, or non-terminating, computation can never do them any good. In the physical

† I am grateful for the support of the National Natural Science Foundation of China (60873034, 61033002, 91318301).

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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 of computation must distinguish between these two kinds of computational object from the outset.

Modern computing environments feature interactions, and interaction leads to non-determinism. In such an environment, a computational object may be capable of producing a result, but at the same time it also has the potential to diverge, and the choice between these two actions is non-deterministic. This type of computational object differs from a terminating computational object and a divergent computational object in both the human and physical worlds. In fact, the situation is far more complicated since non-determinism and divergence can be combined in very complex ways. It is not at all clear if anything can be said about these computational objects.

In the current paper, we set out to study non-deterministic computational objects. Our objective is to answer the following questions:

— What is the right notion of equality for non-deterministic computational objects? A reasonable answer to this question will be the starting point for subsequent investigations.
— What are the structures of non-deterministic computational objects? More specifically, can we characterise the equivalence classes of these objects?
— What are the non-deterministic computational objects defined in a computation or interaction model? Do they differ from one model to another?

In the current paper, we describe the results we have obtained by looking into these questions. Our contributions can be summarised as follows:

— We will present a coherent account of the following interrelated concepts: the Church–Turing thesis; bisimulation; computation; interaction; divergence; non-determinism; computational equality; observational equivalence; and Turing completeness. The equality for the non-deterministic computational objects will emerge from this account.
— We will construct a complete equational system for the equality we have introduced for finite-state computational objects. This system contains a few axiom schemata, and 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, and establish a general definability result.
— We will show that the non-deterministic structure of computation is model independent for a general class of models.

1.1. *Organisation of the paper*

In Section 2, we discuss equality properties for deterministic computation, and in Section 3, we extend these properties to take account of interactions. In Section 4, we characterise finite-state computational objects, and then, in Section 5, we investigate infinite-state computational objects. In Section 6, we establish the model independence property of the non-deterministic computational objects. Finally, in Section 7 we provide some concluding remarks and identify some further research issues.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

2. The 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), which became known as primitive recursive functions after Kleene (1936a), evolved into Herbrand–Gödel's general recursive functions (see Gödel's 1934 lecture notes reprinted in Davis (1965)), although at the time Gödel himself was not very sure that general recursion captured all possible effective calculability. The year 1936 was when the foundations of computation theory began to take shape. Kleene (1936a) provided a characterisation of general recursive functions in terms of primitive recursive functions and the least number operator (the μ-operator). Church (1936) and 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 (which was just called the Church Thesis in Kleene (1952)). It was later pointed out in Kleene (1938) that the thesis also covers the situation of partially defined recursion. Turing remarked in Turing (1937a) that the λ-calculus was more convenient, but what he could not foresee at the time was that, owing to its simplicity, the λ-calculus would play a foundational role for (functional) programming languages (Abramsky 1988). Gödel was reported as saying that Turing machines offer the most convincing formalisation of mechanical procedures (Davis 1965). It came as no surprise that the physical feasibility of Turing machines was exploited in the influential von Neumann architecture.

After the discoveries of the 1930's, other models of computation with different emphases were proposed and proved equivalent to the earlier Gödel, Church, Kleene and Turing models. Post Systems (Post 1943) and Markov Algorithms (Markov 1960) can be viewed as language models, while counter machines (such as Minsky Machines (Minsky 1967) and the Unlimited Register Machines of Shepherdson and Sturgis (1965)) and Random Access Machines (Cook and Reckhow 1973) belong to the class of register machines.

Our understanding of effective calculability can be further enhanced by revealing the hidden depths 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). However, the equivalence proofs developed 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 will need to be a bit more formal about computation models and the translations between them.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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 usually be omitted, is the *one-step computation* relation. Let $\mathfrak{C}$ be the class of all computation models, which will be ranged over by $\mathbb{A}, \mathbb{B}$. We will write $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$.

We 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 $\mathbb{B}$, is 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 $\mathcal{T}(M')$, which requires us to introduce an equivalence on the computational objects of $\mathbb{B}$, which complicates the issue from the outset. However, an elegant way to bypass this problem is to think of $\mathcal{T}$ as a recursive relation rather than as a recursive function. To simplify the following account, we will abuse notation by identifying $C_{\mathbb{A}}$ with the set of 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 finding effectively a computational object M' in $\mathbb{A}$ such that $M \rightarrow^* M'$ and N' is an interpretation of M'. This is usually called the reflection property. These remarks lead us to Definition 2.1, but first, we will fix some notation.

From now on, the set $\mathbf{N}$ will be ranged over by i, j, k. Given a binary relation $\mathcal{S}$, we will use the infix notation $x\mathcal{S}y$ for the membership predicate $(x, y) \in \mathcal{S}$, and write $\mathcal{S}^{-1}$ to denote 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 2.1. A relation $\mathcal{S} \subseteq \mathbf{N} \times \mathbf{N}$ is *recursive* if:

(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

$$d(\mathsf{s}^+) = d(\mathcal{S})$$
$$d(\mathsf{s}^-) = r(\mathcal{S});$$

and

(3) we have

$$\forall x \in d(\mathcal{S}).\mathsf{s}^-(\mathsf{s}^+(x)) = x.$$

By definition, the composition of two recursive relations is a recursive relation. To appreciate the above definition, note that condition (3) 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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

polynomially bounded (van Emde Boas 1990; Wegener 2005). However, we will ignore this *quantitative* aspect of the Church–Turing Thesis in the current paper

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

Definition 2.2. Suppose $\mathbb{A}$ and $\mathbb{B}$ 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:

(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 $M \rightarrow M'$; and
(4) if $M\mathcal{T}N$, 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 $\mathbb{A} \leqslant_e \mathbb{B}$ if there is an effective reduction from $\mathbb{A}$ to $\mathbb{B}$.

As mentioned earlier, this definition makes little sense without condition (2). Condition (3) is the weak bisimulation property (Park 1981; Milner 1989a), which is well known in logic and process theory (Sangiorgi 2009) – we will return to this point later. Conditions (2) and (3) are the fundamental *qualitative* properties of $\mathcal{T}$.

Condition (4), which is 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 non-termination, or divergence as it is normally called, is the simplest model-independent treatment of undefinedness. It is important to realise that if Definition 2.2 is strengthened by adding a quantitative condition saying that the length of a simulating sequence is polynomial in the length of the computation sequence being simulated, then condition (4) is a corollary of condition (3) if $\mathbb{A}$ and $\mathbb{B}$ are both 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 all of the well-known (deterministic) computation models can be regarded as an initial model. However, for the current paper, it will be 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* in the form of a finite sequence of *instructions* $L_1, \ldots, L_k$ with 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;

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

— $DecJump(r_j, s)$ decreases the number in r_j by one if it is not zero, otherwise it jumps to the sth instruction; and
— End stops the execution of the program.

Without loss of generality, we will assume that a program instruction is End if and only if it is the program's last instruction.

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 jth 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: $\forall \mathbb{A} \in \mathfrak{C}. \mathbb{R} \leqslant_{\mathfrak{e}} \mathbb{A}$.

Conditions (3) and (4) of Definition 2.2 are formulated for deterministic computation, but this is not really a restriction for CTT-II since $\mathbb{R}$ is deterministic. We will see in the next section how $\leqslant_e$ can be strengthened so that it applies to all pairs of deterministic/non-deterministic computation models.

3. Church–Turing thesis and process equality

We will now turn to a consideration of non-deterministic computations. In a semantic setting, non-determinism is about state change. Suppose M is a configuration of a non-deterministic Turing machine and that M may evolve non-deterministically in a single step to two distinct configurations M' and M''. If M' and M'' always lead to final configurations with the same numbers in the output tapes, we may think of these two configurations as semantically equivalent. In this case, the non-determinism is fake. True non-determinism comes into the picture when M' and M'' are not semantically equivalent: for example, if it is possible for M' to reach a final configuration with a number on the output tape but M'' can never reach a final configuration with the same output number.

Non-determinism is not about effectiveness: we can never argue in favour of it from the point of view of effective calculability. Non-determinism 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 in the study of process calculi is the theory of observational equivalence. However, before explaining the relationship between the observational theory and the Church–Turing Thesis, we need to give a brief introduction to the $\mathbb{CCS}$ model.

A crucial feature of the definition of $\mathbb{CCS}$ is the notion of a (channel) name. The following is a brief summary of the relevant terminology concerning names and name variables:

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

— 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 various decorated versions. The notation τ stands for an internal action, which is different from any name or 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.

We will sometimes abbreviate a finite sequence of names, such as $c_1, \ldots, c_k$, to $\widetilde{c}$.

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

$$T := S \mid T \mid T' \mid (c)T \mid D(n_1, \ldots, n_k)$$
$$S := \mathbf{0} \mid \ell.T \mid S + S'.$$

For the term $\ell.T$, 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

$$\sum_{1 \leqslant i \leqslant n} \ell_i.T_i$$

for

$$\ell_1.T_1 + \ell_2.T_2 + \ldots + \ell_n.T_n.$$

Note that we are able to omit the brackets in

$$\ell_1.T_1 + \ell_2.T_2 + \ldots + \ell_n.T_n$$

because the operator '+' is commutative and associative. We will use the notation $\sum_{i \in I} \ell_i.T_i$, 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 $\mathbf{0}$, and a trailing $\mathbf{0}$ will often be omitted.

The operator '|' composes two terms into one, allowing the components to interact. In the localisation 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 will assume α-conversion, which means that a local name in a term can be renamed to a fresh name without changing the syntax of the term. We will use α-conversion whenever it is needed 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

$$T\{n_1/x_1, \ldots, n_k/x_k\}.$$

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

We will write A, B, C, D for processes defined by parametric definition. The parameters $x_1, \ldots, x_k$ in equation (1) are bound, and 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 labelled 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

$$\frac{T_0 \xrightarrow{\lambda} T_0'}{T_0 \mid T_1 \xrightarrow{\lambda} T_0' \mid T_1} \qquad \frac{T_0 \xrightarrow{a} T_0' \quad T_1 \xrightarrow{\bar{a}} T_1'}{T_0 \mid T_1 \xrightarrow{\tau} T_0' \mid T_1'}$$

$$\frac{T_1 \xrightarrow{\lambda} T_1'}{T_0 \mid T_1 \xrightarrow{\lambda} T_0 \mid T_1'} \qquad \frac{T_0 \xrightarrow{\bar{a}} T_0' \quad T_1 \xrightarrow{a} T_1'}{T_0 \mid T_1 \xrightarrow{\tau} T_0' \mid T_1'}$$

Localisation

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

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 simultaneously substituting $n_1, \ldots, n_k$ for $x_1, \ldots, x_k$.

If both

$$T \xrightarrow{a} T_1,$$

meaning that T can evolve into T_1 by performing the *external* action a, and

$$T' \xrightarrow{\bar{a}} T_1',$$

meaning that T' can turn into T_1' by performing the *external* co-action $\bar{a}$, then T and T' may interact at the channel a, resulting in an *internal* action

$$T' \mid T \xrightarrow{\tau} T_1' \mid T_1.$$

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

$$\tau.a + \tau.b \xrightarrow{\tau} a$$

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

is a non-deterministic computation step. The reflexive and transitive closure of $\xrightarrow{\tau}$ is denoted by $\Longrightarrow$, and 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 our work here. The grammar of this model is given by

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

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, which we will denote by $\mathbb{CCS}^{\mu}$, there is no need for name variables. The semantics of the fixpoint term is defined by the 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 defined simply by

$$D = T\{D/X\},$$

so we can use the fixpoint notation in $\mathbb{CCS}$. Fu and Lu (2010) showed that $\mathbb{CCS}^{\mu}$ is strictly less expressive than $\mathbb{CCS}$. In fact, while $\mathbb{CCS}$ is Turing complete (Busi *et al.* 2003; Busi *et al.* 2004), $\mathbb{CCS}^{\mu}$ is not (Fu and Lu 2010). Milner used another variant in Milner (1989a), which used dynamic binding and is equivalent to $\mathbb{CCS}$ (Giambiagi *et al.* 2004). We shall use $\mathbb{CCS}^{\mu}$ to characterise finite-state computation and $\mathbb{CCS}$ to study infinite-state computation.

A more restricted recursion is given by the replication operator. A replication term is of the form $!\lambda.T$ with the semantics given by the rule

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

It is clear that $!\lambda.T$ can be defined by

$$\mu X.\lambda.(T \mid X).$$

Had the development of process theory paralleled the development of computation theory, 'reduction' between process calculi would have been introduced at an early stage. However, this is not what happened. The truth is that the equivalence relation, which is a relatively simple concept in computation theory, becomes a most intriguing issue in process theory (Milner 1980), and has remained one of the major concerns throughout its development (Milner 1989a; Hennessy 1988; Baeten and Weijland 1990; Sangiorgi and Walker 2001). This is understandable since we would not know how to define a reduction between two process calculi if we did not know how to define an equivalence on one process calculus. After all, an equivalence can be viewed as a reduction from one process calculus to itself (Fu 2012). For a long time, Milner's weak bisimilarity (Milner 1989a)

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

was the main tool for identifying the interactive behaviours of processes (Milner 1993). The relation was originally defined in terms of both internal and external actions, but Milner and Sangiorgi pointed out later that the bisimulation property for external actions is a derivable property (Milner and Sangiorgi 1992). 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, so we have the following two definitions.

Definition 3.1. A relation $\mathcal{R}$ on processes is *extensional* if both:

$$(c)P \ \mathcal{R} \ (c)Q \quad \text{if} \quad P\mathcal{R}Q$$

and

$$(P \mid P') \ \mathcal{R} \ (Q \mid Q') \quad \text{if} \quad P\mathcal{R}Q \text{ and } P'\mathcal{R}Q'.$$

Definition 3.2. 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 $P\mathcal{R}Q$.

Equipollence is the barbed condition of Milner and Sangiorgi (1992). We will now define Milner's weak bisimilarity in the style of barbed bisimulation (Milner 1989a).

Definition 3.3. The *Milner equality* $\approx$ is the largest relation $\mathcal{R}$ such that:

(1) $\mathcal{R}$ is reflexive.
(2) $\mathcal{R}$ is extensional and equipollent.
(3) $\mathcal{R}$ is a *weak bisimulation.* In other words:
   — 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 $M\mathcal{R}N$, 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 2.2 and Definition 3.3 deserves comment. The totality condition of Definition 2.2 turns into the reflexivity condition of Definition 3.3. This is because the former is a relation on two models, while the latter is a relation on one model. Condition (2) of Definition 2.2, which is an intensional requirement, becomes

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

condition (2) of Definition 3.3, which is an observational requirement. This analogy prompts the following criticisms of $\approx$:

(1) As we have already said, weak bisimulation is essentially a property of deterministic computation, and it does not really fit into a definition that is supposed to take care of the non-determinism caused by interaction.
(2) Condition (4) of Definition 2.2 is completely ignored by $\approx$, and this is a deficiency if process calculi are required to subsume computation models.

Similar doubts may be raised with respect to the equivalence $\approx_{\downarrow}$.

(3) Condition (4) of Definition 2.2 is formulated for deterministic computations, so does it still make sense in the presence of non-determinism?

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 non-deterministic computation

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

is simulated by the two-step non-deterministic 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, so the environment may well intervene at the state $a + \tau.b$ to disrupt the simulation. It is debatable whether (3) should really be seen as a simulation of (2). In computation theory, the distinction between deterministic computations and non-deterministic computations is of fundamental importance, and the issue raised by this example should be properly addressed.

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

Lemma 3.4. Let $\asymp$ be either the weak bisimilarity or the absolute equality to be defined later. If

$$P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_n \asymp P_1,$$

then

$$P_1 \asymp P_2 \asymp \ldots \asymp P_n.$$

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 underlying van Glabbeek and Weijland's branching bisimulation is that a state-changing internal action (a non-deterministic 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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

(a deterministic computation step) has so little consequence that it can be completely ignored. In the following definition, we will omit the adjective 'branching' for brevity.

Definition 3.5. $\mathcal{R}$ 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

$$\exists Q'.Q'' \xrightarrow{\tau} Q'\mathcal{R}^{-1}P'.$$

We will now turn to the second criticism of $\approx$, which is that the equivalence $\approx$ does not differentiate between

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

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

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

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

So 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 non-deterministic 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, while the latter keeps offering the environment the possibility of interaction at channel d.

However, it is not difficult to see how we can modify the termination preservation condition to take non-determinism 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. Now, if (7) contains an infinite number of non-deterministic computation steps, the bisimulation property guarantees that (7) is bisimulated by an infinite computation. However, bisimulation fails to achieve this if (7) contains

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

only a finite number of non-deterministic computation steps. The additional requirement is captured by the condition introduced in the following definition.

Definition 3.6. 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 3.6 was introduced in Priese (1978), but was rediscovered in different contexts in van Glabbeek *et al.* (2009) and Fu and Lu (2010).

We can now, at last, give our model-independent definition of process equality. For this, we assume that all process models have both the concurrent composition operator and the restriction operator.

Definition 3.7. The *absolute equality* $=_{\mathbb{M}}$ of a process model $\mathbb{M}$ is the largest binary relation on $\mathbb{M}$ processes such that:

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

We shall often omit the subscript in $=_{\mathbb{M}}$. The definition of absolute equality is uniform for all models – further motivation for the use of absolute equality can be found in Fu (2012).

An interesting equality concerning unobservable processes is given by the following lemma.

Lemma 3.8. $P \mid \Omega = \Omega$ for every unobservable process P.

Proof. Let $\mathcal{R}$ be

$$\{(P_0, P_1) \mid \forall i \in \{0,1\} . P_i \equiv \Omega \vee (P \mid \Omega \Longrightarrow P_i \text{ and } P \text{ is unobservable})\}$$

and let $\mathcal{R}^\circ$ be the least relation satisfying:

(i) $\mathcal{R} \subseteq \mathcal{R}^\circ$;
(ii) $\mathcal{R}^\circ$ is reflexive;
(iii) if $P \mathcal{R}^\circ Q$, then

$$(P \mid O) \mathcal{R}^\circ (Q \mid O)$$

and

$$(O \mid P) \mathcal{R}^\circ (O \mid Q)$$

for every process O; and

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

(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 give a formal definition of the terminology we have been using informally so far:

— We write

$$T \to T'$$

if

$$T \xrightarrow{\tau} T' = T,$$

and we say that $T \to T'$ is a *deterministic computation* step since T and T' must both have the same input/output capacities.

— We write

$$T \xrightarrow{\iota} T'$$

if

$$T \xrightarrow{\tau} T' \neq T,$$

and we say that $T \xrightarrow{\iota} T'$ is a *non-deterministic computation* step because either T and T' have different input/output behaviours or they have different divergent behaviours.

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 for deterministic computation models.

Returning to the examples given in equations (5) and (6), we have $D_1 \approx_{\downarrow} D_2$, but $D_1 \neq D_2$ since D_1 can perform an infinite sequence of deterministic computation steps but D_2 cannot, and it is the codivergence condition that distinguishes between them.

Given Definition 3.7, we can now define effective translations between computation models that admit non-determinism.

Definition 3.9. A relation $\mathcal{R}$ from a model $\mathbb{A}$ to a model $\mathbb{B}$ is an *effective sub-bisimilarity* if:

(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 sub-bisimilarity from $\mathbb{A}$ to $\mathbb{B}$.

We base the terminology 'sub-bisimilarity' on 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 more generally than $\leqslant_e$: it is obvious that $\mathbb{R} \sqsubseteq_e \mathbb{A}$ implies $\mathbb{R} \leqslant_e \mathbb{A}$. So CTT-II can be strengthened in terms of the more informative relation $\sqsubseteq_e$.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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

CTT-III applies to both computation 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 us 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 our work here – apart from the argument for the recursiveness, the proof is taken from Busi *et al.* (2003).

Proposition 3.10. $\mathbb{R} \sqsubseteq_e \mathbb{CCS}$.

Proof. The jth register r_j with the initial value 0 is interpreted by Z_j defined by

$$Z_j = zero_j.Z_j + inc_j.(c)(S_j(c) \mid c.Z_j) \tag{8}$$

$$S_j(x) = dec_j.\overline{x} + inc_j.(c)(S_j(c) \mid c.S_j(x)). \tag{9}$$

The value of the register can be increased and decreased by interacting at the channels inc_j and dec_j, respectively. The standard form of the interpretation of r_j with value i is the process

$$(c_{i-1} \dots c_0)(S_j(c_{i-1}) \mid c_{i-1}.S_j(c_{i-2}) \mid \dots \mid c_2.S_j(c_1) \mid c_1.S_j(c_0) \mid c_0.Z_j).$$

Suppose $L_1, L_2, \dots, L_k$ is the program of a CM. For each $i \in \{1, \dots, k-1\}$, the instruction L_i is interpreted by

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

The final instruction, which is *End*, is simply interpreted as $\mathbf{0}$.

To achieve recursiveness, we use a fixed countable set of names $\{c_0, c_1, c_2, \dots\}$. We also need the following commutative and associative rewriting system:

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

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

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

where:

— $L_1, \dots, L_k$ is the program of CM;
— $D(L_1), \dots, D(L_k)$ are the parametric definitions interpreting the instructions;
— I_j is the interpretation of the current instruction; and
— $N_1, \dots, N_{k'}$ are the normal forms interpreting the values of the registers of CM.

Let $\mathcal{R}$ be the set of the pairs of the form

$$\langle \mathsf{CM}, \langle N'_1, \dots, N'_{k'}, I_j, D(L_1), \dots, D(L_k) \rangle \rangle$$

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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) Whether a $\mathbb{CCS}$ process is the interpretation of a CM configuration is decidable.
(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 us 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 behaviour must exist, and we may refer to it without explicitly defining it. We can 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, denoted by

$$\mathsf{C}^{\mathsf{f}}(x, u, y, v, z, w).A,$$

such that

$$F \stackrel{\text{def}}{=} \mathsf{C}^{\mathsf{f}}(in, end, out, stop, dec, zero).A$$

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

$$F \underbrace{\rightarrow^* \xrightarrow{in} \ldots \rightarrow^* \xrightarrow{in}}_{k} \xrightarrow{end} \underbrace{\rightarrow^* \xrightarrow{out} \ldots \rightarrow^* \xrightarrow{out}}_{\mathsf{f}(k)} \xrightarrow{stop} = A \mid [\![k]\!]^{zero}_{dec},$$

where

$$[\![k]\!]^{zero}_{dec} \equiv \underbrace{\overline{dec}.\cdots.\overline{dec}}_{k}.\overline{zero}.$$

For those who are not yet confident in the interactive version of Turing completeness, we point out that the parametric definition $\mathsf{C}^{\mathsf{f}}(x, u, y, v, z, w).A$ is given formally by

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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

where

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

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 $\overline{a}$. Summarising, the process F inputs the number k at channels in, end, and then calculates $\mathsf{f}(k)$. If $\mathsf{f}(k)$ is undefined, it loops forever, otherwise, it outputs the number $\mathsf{f}(k)$ at channels $out, stop$ and copies the number k to channel $dec, zero$ before it starts A. The number k is retained because later calculation may well need it. It is easy to see how to generalise 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 can now 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 now need to promote this soundness condition from the level of computation to the level of interaction.

Definition 3.11. 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 with Definitions 2.2 and 3.7.

Definition 3.12. A relation from the set of $\mathbb{M}$-processes to the set of $\mathbb{N}$-processes is a *sub-bisimilarity* if:

(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 sub-bisimilarity 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 the current paper, there are only a few simple references to $\sqsubseteq$, so we shall not elaborate any further on it here, but see Fu (2012) for a full exposition on the expressiveness relation.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

We will now fix the class $\mathfrak{M}$ of interaction models we are concerned with in the current paper, where our running theme 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 the models we are interested in:

— There is an effective bijection between $\mathbf{N}$ and the set of $\mathbb{M}$-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 $P \stackrel{\tau}{\longrightarrow} P'$.

These effective conditions have been used to study structural operational semantics, giving rise to effective operational semantics (Vaandrager 1993). Variants of these conditions have also been proposed in a number of papers, including Baeten *et al.* (1987), de Simone (1985), Darondeau (1990) and Bloom *et al.* (1995).

Our interaction models are sufficiently expressive that:

(i) they are Turing complete; and

(ii) they are at least as expressive as $\mathbb{CCS}$.

By Proposition 3.10, (i) is subsumed by (ii), so we get the following definition.

Definition 3.13. An interaction model $\mathbb{M}$ for which the three statements above are true is said to be a *Turing–Milner model* if $\mathbb{CCS} \sqsubseteq \mathbb{M}$.

In a Turing–Milner model, the execution tree of an unobservable process can be generated algorithmically. The finite branching versions of the well-known complete process calculi are Turing–Milner models. These include both the π-calculus (Milner *et al.* 1992a; Milner *et al.* 1992b; 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 tricky. However, we know that $\mathbb{CCS}^{\mu}$ is not a Turing–Milner model since it is not even complete (Fu 2012).

From now on, we will write $\mathfrak{M}$ to denote 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 to give a condensed account of a journey starting from the original formulation of the Church–Turing Thesis, through Milner and Park's notion of weak bisimulation and van Glabbeek and Weijland's discovery of the branching bisimulation, until we have arrived at a technical formalisation of the thesis. Our journey was guided by the fact that it is within the theory of interaction that the full picture of non-deterministic computations can be revealed. It is against this background that Definition 3.7 emerges as a canonical equality for both interactive and computational objects. The purpose of the current paper is to begin an investigation into the rich structures of non-deterministic computation revealed by this equality.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

4. Finite-state computation

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

$$T := X \mid S \mid \Delta(T) \mid \mu X.T$$
$$S := \mathbf{0} \mid \tau.T \mid S + S.$$

The Δ operator was introduced by Hennessy and Milner (see Hennessy (1981), where it was called a delay operator and denoted by δ) and 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 $T \to T'$, that is

$$T \xrightarrow{\tau} T' = T,$$

and the transition $T \xrightarrow{\tau} T'$ is not caused by a self-loop. If there is no T' such that $T \to T'$, we write $T \not\rightsquigarrow$. The notation $T \not\to$ is defined similarly.

The simplest terminating computational object is of course $\mathbf{0}$, and the simplest divergent computational object is Ω, which was defined in (4). All other finite-state computational objects have the potential both to terminate and to diverge. The execution of $\Delta(\tau)$ can be described by the labelled tree

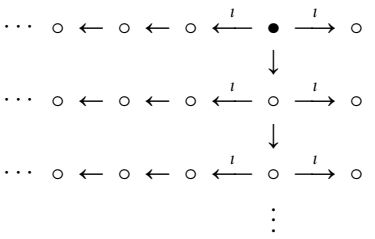
$$\begin{array}{l} \bullet \xrightarrow{\iota} \circ \\ \downarrow \\ \circ \xrightarrow{\iota} \circ \\ \downarrow \\ \circ \xrightarrow{\iota} \circ \\ \vdots \end{array}$$

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

$$\cdots \circ \leftarrow \circ \leftarrow \circ \xleftarrow{\iota} \bullet \xrightarrow{\iota} \circ$$

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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



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

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

$$\Upsilon_0 = \mathbf{0}$$
$$\Upsilon_1 = \Delta(\tau.\mathbf{0})$$
$$\Upsilon_2 = \tau.\mathbf{0} + \tau.\Omega.$$

Starting from Υ_3, the sequence is defined recursively by the following equations, 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 4.1. $\forall i > 0. \forall j < i.\ \Upsilon_j \neq \Upsilon_i$.

Proof. We use a natural induction 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 $k \geqslant 3$.

— Suppose $k \leqslant 2i$.
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, the induction hypothesis means

$$\Upsilon_{2i+1} \xrightarrow{\tau} \Upsilon_{2i}$$

cannot be bisimulated by Υ_k. So the process Υ_{2i+1} is not equal to any Υ_k with $k \leqslant 2i$.

— Suppose $k \leqslant 2i + 1$.
The action

$$\Upsilon_{2i+2} \xrightarrow{\iota} \Omega$$

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

cannot be matched by any action of Υ_k whenever k is odd. If k is even, the induction hypothesis means

$$\Upsilon_{2i+2} \xrightarrow{\tau} \Upsilon_{2i+1}$$

cannot be bisimulated by Υ_k. So the process Υ_{2i+2} is not equal to any Υ_k with $k \leqslant 2i+1$. □

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 4.2. 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'.

It is obvious that $\sim \subsetneq = \subsetneq \approx$.

An equality on processes can be extended to an equality on terms in the standard manner. For example, $T = T'$ if

$$T\{P_1/X_1, \ldots, P_n/X_n\} = T'\{P_1/X_1, \ldots, P_n/X_n\}$$

for all processes $P_1, \ldots, P_n$.

We will 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 due to Bloom and Ésik are also valid with respect to the Milner–Park equality (Bloom and Ésik 1994):

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

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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

If we think 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:

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

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

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

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

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

where 'X is guarded in T' means that every occurrence of X in T is guarded by at least one non-τ prefix.

The equalities (13)–(18) have all been used as axioms in equational systems for finite-state behaviours. However, if we intend to construct an equational proof system for the absolute equality $=$ on the finite-state computational objects, none of (16), (17) or (18) will be useful 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.

We will now turn to the algebraic properties of divergence. Milner's approach to obtaining complete axiom systems for finite-state behaviours relies heavily on the fixpoint induction rule, but if we try to apply it to tackle the divergence problem for the finite-state processes, we 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 of reasoning about divergence using fixpoint induction.

Lohrey *et al.* (2002) and Lohrey *et al.* (2005) provided an ingenious solution that bypasses this dilemma. Their key observation was 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*. So an induced operator that helps us rewrite all τ-loops to self-loops would be very useful. We could use the unguarded choice operator to define it by

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

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

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

However, we cannot use (20) to define Δ for our work here because we do not use the general choice operator, which is why we introduced Δ as a primitive operator. The

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

equalities studied in Lohrey *et al.* (2002) and Lohrey *et al.* (2005) are all weaker than the absolute equality, but some of their laws are still valid for $=$, for example,

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

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

Again, these laws make use of the unguarded choice.

To summarise, the issues we will need to resolve in order to produce an equational system are as follows:

(1) Most of the laws concerning the μ-operator and the Δ-operator are no longer available, so we need to come up with some alternatives.
(2) We need to find an equational replacement for fixpoint induction.

Fixpoint induction allows us to prove the equality between a short term and a very long term, for example

$$\mu X.a.X = \mu X.\underbrace{a\ldots a}_{k \text{ times}}.X$$

where $k > 0$. Sewell (1994) and Sewell (1997) proved that no pure equational system is strong enough to derive this equality for all $k > 0$, and we will give a similar non-existence result in Section 4.3. Our solution to this problem is to use axiom schemata, and to do this, we will need to use C-contexts.

Definition 4.3. A *C-context* is either $[_]$, or one of

$$\tau.C[_]+S$$
$$S+\tau.C[_]$$
$$\Delta(C[_])$$
$$\mu X.C[_]$$

where $C[_]$ is a C-context.

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

Lemma 4.4. The following equations are valid:

$$\mu X.C[\tau.(\tau.X+S_1)+S_2] = \mu X.C[\tau.X+S_1+S_2]$$
$$\mu X.C[\Delta(X)] = \mu X.C[\tau.X]$$
$$\mu X.C[\Delta(\tau.X+S)] = \mu X.C[\tau.X+S].$$

Proof. All three equalities can be established using the Computation Lemma. □

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 trade-off is that the absolute

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

equality is no longer a congruence. It is well known that the binary choice operator does not preserve $=$. We have $\mathbf{0} = \tau$, but

$$\mathbf{0} + \tau.\Omega \neq \tau + \tau.\Omega.$$

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 4.5. P and Q are congruent, denoted by $P \asymp Q$, 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

$$P \equiv \sum_{i \in I} \tau.T_i$$
$$Q \equiv \sum_{j \in J} \tau.T_j$$

where $\equiv$ is the syntactic equality, and neither P nor Q contains any occurrences of the μ-operator. If

$$Q = P \xrightarrow{l} P'$$

and $Q' \neq P'$ for all Q' such that $Q \xrightarrow{\tau} Q'$, then it is easy to see that

$$P + \tau.\Upsilon_k \neq Q + \tau.\Upsilon_k$$

for some large enough k. This explains why clause (1) of Definition 4.5 cannot be replaced by

(1′) If $P \xrightarrow{\tau} P'$, then $Q \rightarrow^* \xrightarrow{\tau} Q' = P'$ for some Q'.

The proof of the next proposition is standard.

Proposition 4.6. The equivalence $\asymp$ is the largest congruence contained in $=$.

The proof system AC for finite-state computational objects consists of the axioms defined in Figure 1. We write $AC \vdash T = T'$ to indicate the fact that the equality $T = T'$ 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 (1989) by restricting to the τ-prefix. The T law is a special case of Milner's first tau law. A special case of C is

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

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 with the laws (21) and

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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

(22) given by Lohrey *et al.* (2002) and Lohrey *et al.* (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, which appeared more than 30 years ago in Hennessy (1981), is weaker than the following axiom due to Lohrey, D'Argenio and Hermanns:

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

The equality (24) fails the codivergence condition.

Proposition 4.7. 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 it is clear that F3 also holds in this case. F4, F5 and F6 are valid by Lemma 4.4, and D1 and D2 are in fact true for the Milner–Park equality. □

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, and implies the equality

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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press



Fig. 2. Two D-graphs

4.2.1. *D-normal form.* The finite-state computational objects can be viewed as graphs. Conversely, finite graphs of a certain type can be coded up by finite-state computational objects. It is then trivial to generalise the finite graphs that represent the finite-state computational objects to infinite graphs.

Definition 4.8. A directed graph is a *D-graph* if the following conditions 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 Figure 2 are D-graphs. A root is indicated by '•'. If the root self-loops, a '↻' is placed immediately below the '•'. If a node is not a root, it is indicated by '○'; and if it self-loops, it is simply indicated by '↻'.

Definition 4.9. 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 will begin 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 ith node has no outgoing edges, we have $X_i = \mathbf{0}$, and if the node has a self-loop, we introduce the equation $X_i = \Delta(\mathbf{0})$. If the ith node has k children $X_{m_1}, \ldots, X_{m_k}$, then the ith equation is

$$X_i = \tau.X_{m_1} + \ldots + \tau.X_{m_k}.$$

If the ith node also has a self-loop, the ith equation is

$$X_i = \Delta(\tau.X_{m_1} + \ldots + \tau.X_{m_k}).$$

It is easy to construct solutions to the equation system using the standard method (Milner 1984). The solution to X_0 is represented by the D-graph.

We will now describe a procedure that converts a finite computational object to one that corresponds to a D-graph. To do this, we will introduce D-normal forms.

Definition 4.10. A term T is a *D-normal form*, or *D-nf*, if:

(1) T is a variable.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

(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.

For example, the finite state

$$\mu X.\mu Y.(\tau.X + \tau.Y)$$

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. So 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.
— Ω is the D-graph with only one node and the self-loop.
— The Σ-process

$$\sum_{1\leqslant i\leqslant n} \tau.T_i$$

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

Proof. According to the structural definition of a D-nf, a subterm of a D-nf is a D-nf, and if T and T' are in a 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 for these cases.
— $T \equiv \tau.T'$:
We just apply the induction hypothesis to T'.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

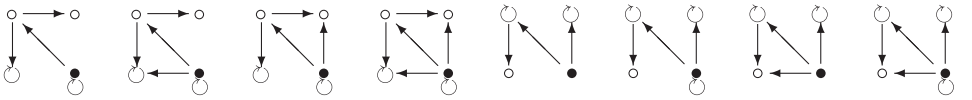


Fig. 3. C-Graphs with four nodes

— $T \equiv \sum_{i\in I} \tau.T_i$:
We apply the induction hypothesis to each T_i. Note that we can apply S1 to remove a 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 $\Delta(S)$, we apply F6 to remove the Δ-operator.
  – Finally, we use F5 to remove the Δ-operator in all occurrences of $\Delta(X)$. □

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 3.6 and bisimilar in the sense of Definition 3.5. The D-graph on the left-hand side in Figure 2 has two pairs of equal nodes while the D-graph on the right-hand side has one pair of equal nodes. The structure of computations is given by the D-graphs containing no equal nodes.

Definition 4.12. A *C-graph* is a D-graph in which no two nodes are equal.

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

It follows from the 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, which will be ranged over by $\mathfrak{f}, \mathfrak{g}, \mathfrak{h}, \mathfrak{i}$. We will write $\mathfrak{f}$ for a finite C-graph and $\mathfrak{i}$ for 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 non-empty finite set for each $k > 0$.

Lemma 4.13. 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, we can construct a codivergent bisimulation by pairing all the corresponding nodes.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

Conversely, suppose the roots of the graphs are equal. Given any node N in $\mathfrak{g}_0$, there is a unique path from the root to N. Using the bisimulation property, we 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 $\mathfrak{g}_1$. It is easy to see by induction that it is an isomorphic map. □

According to Lemma 4.13, the C-graph representation of an unobservable process P is unique. Consequently, we can talk about equality between C-graphs. We will write $\mathfrak{c}(P)$ for this unique C-graph.

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

Definition 4.14. A D-nf T is a *C-normal form*, or *C-nf*, if:

(1) T does not contain any occurrences of the μ-operator; and
(2) one of the following statements is true:
   (a) either $T \equiv X$ 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;
      — $\forall j,k \in I.j \neq k \Rightarrow T_j \neq T_k$; 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) and (2b) ensure that $T \not\rightarrow$, which means that T contains no loops other than self-loops. If T_1 and 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 4.15. Suppose T_1, T_2 are C-nf's and $T_1 = T_2$. Then:

(1) If both T_1 and T_2 are Σ-terms,

$$AC \vdash \tau.T_1 = \tau.T_2.$$

(2) If both T_1 and T_2 are Δ-terms,

$$AC \vdash T_1 = T_2.$$

Proof. The proof is carried out by simultaneous structural induction. Since a Δ-term loops but 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.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

— Suppose

$$T_1 \equiv \sum_{i\in I} \tau.T_1^i$$
$$T_2 \equiv \sum_{j\in J} \tau.T_2^j$$

with $I \neq \varnothing \neq J$. 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,

$$AC \vdash \tau.T_1^i = \tau.T_2^j,$$

so

$$AC \vdash T_2 + \tau.T_1^i = T_2 + \tau.T_2^j = T_2.$$

We conclude by induction and symmetry that

$$AC \vdash T_1 = T_1 + T_2 = T_2.$$

— Suppose

$$T_1 \equiv \Delta\left(\sum_{i\in I} \tau.T_1^i\right)$$
$$T_2 \equiv \Delta\left(\sum_{j\in J} \tau.T_2^j\right).$$

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

$$AC \vdash T_1 = T_2,$$

which completes the proof. □

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

Proposition 4.16. The following statements are valid:

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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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

Proof. Note first that we can remove all occurrences of μ-operators in a term 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.
— If X appears in T and $T \not\equiv X$, then:
  – 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, and we can remove all such occurrences as follows:
    • 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 or Δ, apply F3.

  Note that the transformation does not create any new occurrences of X.

It follows from the observation and the proof of Lemma 4.11 that we may assume that T is a D-normal form containing no occurrences of the μ-operator. We now argue by structural induction:

— $T \equiv \mathbf{0}$ or $T \equiv X$:
  There is nothing to prove in these cases.
— $T \equiv \sum_{i \in I} \tau.T_i$:
  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 4.15. So we may remove one of them using S4. Without loss of generality, we assume that for all $i, i' \in I$, we have $T'_i \neq T'_{i'}$ 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 to consider:

  – 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

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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

by Proposition 4.15. 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, we 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 from 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

$$\Delta\left(\sum_{i\in I} \tau.T_i\right) = T_k$$

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

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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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. □

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

Theorem 4.17. $T' \asymp T''$ 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:

— T' is $\mathbf{0}$:
Then T'' must be $\mathbf{0}$.

— T' is X:
Then T'' must be X.

— T', T'' are Δ-terms:
Then $AC \vdash T' = T''$ by Propositions 4.15 and 4.16.

— $T' \equiv \sum_{i\in I} \tau.T_i \asymp \Delta(S) \equiv T''$:
By Proposition 4.16 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 non-empty, 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 by 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, we must have $T_k \asymp \Delta(S)$. If $S \equiv \mathbf{0}$, we can use (27) to complete the proof. If S is not $\mathbf{0}$, it must be a proper choice term. By a bisimulation argument and induction, it is easy to see that

$$AC \vdash T_k = \tau.\Delta(S) + S = \Delta(S).$$

— T', T'' are both Σ-terms:
Then $AC \vdash T' = T''$ follows from Propositions 4.15 and 4.16 and the C law. □

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

It is easy to verify that

$$\mu X.\Delta(T) \asymp \Delta(\mu X.T\{\Delta(X)/X\}).$$

So Theorem 4.17 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. *Non-axiomatisability*

One may feel a little unhappy about the C-contexts appearing in the axiomatic system defined in Figure 1. However, the use of such contexts is unavoidable. The situation is reminiscent of Sewell's remarkable result on the non-existence of a finite purely equational system for finite-state processes (Sewell 1994; Sewell 1997). In this section, we shall establish a non-existence result for *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, because schematic axioms like F4, F5 and F6 are banned.

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

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

is not provable in the system.

Theorem 4.18. There does not exist any sound and complete finite pure axiom system for $\asymp$ on the finite-state terms.

Proof. For the purposes of this proof, we will 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

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

We will prove by induction on the derivation that T' must be a μ-term whose size is at least k:

— If (33) is derived by the transitive rule, we just 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 since otherwise we could instantiate the X on the left-hand side by a term with sufficiently many nested τ prefixes that a computation

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

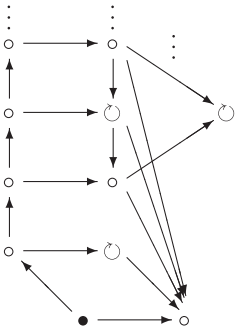


Fig. 4. An infinite C-graph

sequence from the left-hand side cannot be matched by any computational sequences on the right-hand side. We now claim that every occurrence of X in T_c is prefixed by τ at least k times. If this were not the case, then

$$T_c \underbrace{\xrightarrow{\tau} \ldots \xrightarrow{\tau}}_{j \text{ times}} X \tag{34}$$

for some $j < k$. So we 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 $j' \leqslant j$ such that

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

It is apparent that (35) is a contradiction since 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, and 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 then follows by induction that (32) is not provable in AC_p. □

Theorem 4.18 adds considerable weight to the system defined in Figure 1. As a further study, we could carry out an investigation of the system using the approach used in Mendler and Lüttgen (2010).

5. Infinite-state computation

In this section, we will consider infinite C-graphs. Our first infinite C-graph is illustrated in Figure 4. Note 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 necessary, but once it takes a horizontal move, it reaches a node with a finite number of descendants. 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, essentially, they represent the sequence $\Upsilon_0, \Upsilon_1, \Upsilon_2, \ldots$. A node on

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

the left chain is unequal to a node on the right chain because the former can engage in an infinite computation composed of non-deterministic steps, whereas the latter cannot. As this example shows, the nodes in an infinite C-graph can be classified into two types: those that admit an infinite sequence of non-deterministic computations and those that do not.

Definition 5.1. 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 $k \geqslant 0$.

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

Lemma 5.2. If $\mathfrak{i}$ is an infinite C-graph and none of its nodes 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 non-empty. 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. □

It is obvious that 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 this question, we will explain a general construction of an infinite C-graph, which is demonstrated in Figure 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 kth vertical arrow points to the root of f_k. As it stands, the diagram of Figure 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. To do this, we will name these nodes $N_0, N_1, N_2, N_3, \ldots$. For each $i \geqslant 0$, 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 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 4.13. 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 $k \geqslant 0$, 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 non-deterministic computation, but the latter does not.

Theorem 5.3. An infinite C-graph has an infinite number of extremal nodes.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

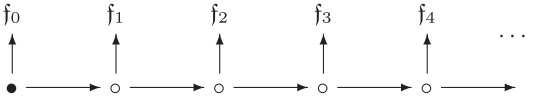


Fig. 5. Construction of a general infinite C-graph

Proof. In order to show a contradiction, we suppose there is an infinite C-graph $\mathfrak{i}$ containing a finite number of extremal nodes. Hence, none of the nodes of $\mathfrak{i}$ has an infinite number of outgoing edges pointing to the extremal nodes. It follows from Lemma 5.2 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 $\mathfrak{f}_1, \mathfrak{f}_2, \ldots, \mathfrak{f}_k$. We will say that $\mathfrak{f}_i$, where $1 \leqslant i \leqslant k$, *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 $\mathfrak{f}_i$. We now define the subgraphs $\mathfrak{i}_0^\omega, \mathfrak{i}_1^\omega, \ldots, \mathfrak{i}_k^\omega$ of $\mathfrak{i}^\omega$ successively by the following induction:

— Let $\mathfrak{i}_0^\omega$ be $\mathfrak{i}^\omega$.
— For $i \in \{1, \ldots, k\}$, if there is some node N_i in $\mathfrak{i}_{i-1}^\omega$ to which $\mathfrak{f}_i$ does not belong, then let $\mathfrak{i}_i^\omega$ be the subgraph of $\mathfrak{i}_{i-1}^\omega$ rooted by N_i; otherwise let $\mathfrak{i}_i^\omega$ be $\mathfrak{i}_{i-1}^\omega$.

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

5.1. *Definability*

We will now turn to the issue of definability and ask if the behaviour of the infinite C-graph in Figure 4 is definable in $\mathbb{CCS}$. The answer is yes. Consider the process *Centipeda* defined as follows:

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

where

$$Cp = \tau.\Upsilon_0 + \tau.(\tau.\Upsilon_1 + \tau.(\overline{even} \mid !even.\overline{inc}.\overline{odd} \mid !odd.\overline{inc}.\overline{even}))$$

$$Cnt = inc.(d)(A(d) \mid d) \tag{36}$$

$$A(x) = dec.\overline{x} + inc.(d)(A(d) \mid d.A(x)) \tag{37}$$

$$O = \mu X.(\tau.X + \tau + \overline{dec}.E) \tag{38}$$

$$E = \tau + \tau.\Omega + \overline{dec}.O. \tag{39}$$

The component Cp is the main process of *Centipeda*, and 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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

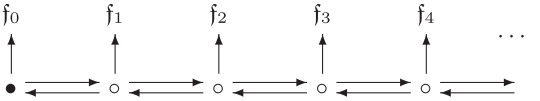


Fig. 6. Construction of a general infinite branching C-graph

decrement. 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 may be definable. We will now substantiate this intuition.

Definition 5.4. 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 numbers 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 non-determinism. However, they can be used to indicate the richness of the non-deterministic structures of C-graphs.

Theorem 5.5. 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:

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

where E and O are defined in (38) and (39), respectively, and F is defined in terms of the general construction described on page 1310. The role of *Reo* is to generate a random 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. It is clear that the process Gr generates the set R. □

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. This is why a D-graph is not required to be finite branching. Figure 6 shows a typical construction of a finite branching D-graph equivalent to an

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

infinite branching C-graph. By the Computation Lemma, all the nodes in

$$\bullet \rightleftarrows \circ \leftrightarrows \circ \leftrightarrows \ldots$$

are equal.

Theorem 5.6. 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 such that, for each $j \geqslant 0$, there are an 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 1310. Let Qng be

$$(c)(\overline{c} \mid !c.\overline{in}.\overline{c} \mid c.\overline{end}).$$

The process Qng is a quasi-number generator since it may never stop. Let Cq be the process

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

Finally, the process Fr is defined by

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

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 $i \geqslant 0$ 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 5.3, it is easy to prove that all the nodes in Tr are equal. This means that the root of $\mathfrak{c}(Fr)$ points to the root of every graph in $\Upsilon_{k_0}, \Upsilon_{k_1}, \Upsilon_{k_2}, \ldots$, so we can conclude that the root of $\mathfrak{c}(Fr)$ is infinite branching. □

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

Theorem 5.7. Every recursively enumerable set can be generated by an unobservable $\mathbb{CCS}$ process.

6. Church–Turing thesis and non-determinism

We can now ask whether the unobservable processes defined in one model differ from those definable in another model. More specifically, is there a C-graph definable in the

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

π-calculus (Milner *et al.* 1992a; Milner *et al.* 1992b) that is not definable in $\mathbb{CCS}$? It is tempting to start the investigation by trying to answer the second question. It turns out, however, that the technique allowing us to answer the specific question also helps us answer the general question. The following theorem tells us that the unobservable processes definable in one Turing–Milner model are definable in another Turing–Milner model.

Theorem 6.1. $\forall \mathbb{M} \in \mathfrak{M}. \forall P \in \mathbb{M}. (P \not\Downarrow \Rightarrow \exists Q \in \mathbb{CCS}. (Q \not\Downarrow \wedge Q = P))$.

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 as follows:

— 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 1310.

— 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 ith 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 1310, 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;
- b^+, e^+, c, f are used to 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 left unchanged.

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

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

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

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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

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 $i = 0$, the process halts. If $i > 0$, the process executes the while-command defined by the two replication processes. In the jth loop of the execution, where $0 < j \leqslant i$, the Gödel index k_j of the jth child of P is calculated and stored at the local names a, d. By the end of the execution, we have

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

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, which simulate all the one-step computations of the $\mathbb{M}$-process P. □

Theorem 6.1 should be interpreted as saying that, from the perspective of the Church–Turing Thesis, the non-deterministic structure of computation is model independent. This result tells us a number of things:

(1) If we want to study the non-determinism of computation, we can choose any Turing–Milner model without losing any generality.
(2) One Turing–Milner model differs from another Turing–Milner model by having different sets of external actions. Hence, one model is more expressive than another if its external actions are more expressive than those of the other.
(3) The approach to defining the universal equality $=$ and the universal expressiveness relation $\sqsubseteq$ in terms of internal actions (computations) is justified – Theorem 6.1 can be viewed as a formal justification of Definitions 3.7 and 3.12.

Our careful choice of the equality has paid off. The non-deterministic structures of computation revealed by the absolute equality are rich, and at the same time remain invariant in a general class of models. Such a theory would have been impossible had we used the weak bisimilarity or termination preserving weak bisimilarity since weak bisimilarity identifies all computational objects and the termination preserving weak bisimilarity distinguishes the three computational objects $\mathbf{0}$, Ω and $\tau.\mathbf{0} + \tau.\Omega$, but identifies any other computational object to one of these three.

Finally, we should mention that investigations similar to those in Section 5 and the current section have also been carried out for effective operational semantics (de Simone 1984; de Simone 1985; Vaandrager 1993), and further studies are required to determine if there is any connection between that work and the work presented here.

7. Remarks

Our understanding of non-determinism for both finite-state and infinite-state computations has been improved. There is an algorithm for checking if a finite D-graph is a C-graph, based on which, we can design a brute force algorithm to enumerate all the elements of $\mathfrak{F}_k$ for every $k > 0$. However, we do not currently know of any closed formula to calculate the size of $\mathfrak{F}_k$. Although this problem belongs in the field of enumerative combinatorics, trying to solve it will definitely improve our understanding of finite-state computations. The problem of checking the equivalence of two unobservable $\mathbb{CCS}$ processes is probably beyond the arithmetical hierarchy (Srba 2004a). Checking the

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

equality between an unobservable $\mathbb{CCS}$ process and a finite C-graph poses a much easier 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 required to clear up these equivalence checking issues. Another problem that also deserves further investigation is a complete characterisation of the C-graphs definable in Turing–Milner models, and given Theorem 6.1, it should be worth the effort of working out the answer to this question.

Acknowledgments

I would like to thank Chaodong He for raising my interest in the unobservable processes, and am grateful to the members of BASICS for the interest they have shown. I am also considerably indebted to the two anonymous referees and the MSCS editorial staff for their constructive criticisms, suggestions and language assistance, which have helped to improve the quality of the paper tremendously.

References

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

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

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

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

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. Unpublished paper.

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

Busi, N., Gabbrielli, M. and Zavattaro, G. (2003) Replication vs Recursive Definitions in Channel Based Calculi. In: Proceedings ICALP'03. *Springer-Verlag 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: Proceedings ICALP'04. *Springer-Verlag 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 the 20th Century. In: Gabbay, D. and Woods, J. (eds.) *Handbook of the History of Logic, Volume 5: Logic from Russell 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.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

Darondeau, P. (1990) Concurrency and Computability. In: Semantics of Systems of Concurrent Processes: Proceedings LITP Spring School on Theoretical Computer Science. *Springer-Verlag 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: van Leeuwen, J. (ed.) *Handbook of Theoretical Computer Science: Algorithm and Complexity, volume A*, Elsevier, 65–116.

Fu, Y. (2012) Theory of Interaction. Unpublished paper.

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

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

Giambiagi, P., Schneider, G. and Valencia, F. (2004) On the Expressiveness of Infinite Behavior and Name Scoping in Process Calculi. In: FOSSACS 2004. *Springer-Verlag 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. (1931) 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.

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. (1936a) General Recursive Functions of Natural Numbers. *Mathematicsche Annalen* **112** 727–742.

Kleene, S. (1936b) 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: Proceedings ICALP 2002. *Springer-Verlag Lecture Notes in Computer Science* **2380** 585–596.

Lohrey, M., D'Argenio, P. and Hermanns, H. (2005) Axiomatising Divergence. *Information and Computation* **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? *Information and Computation* **208** 634–651.

Milner, R. (1980) A Calculus of Communicating Systems. *Springer-Verlag 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. (1989a) *Communication and Concurrency*, Prentice Hall.

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

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press

Milner, R. (1992) Functions as Processes. *Mathematical Structures in Computer Science* **2** 119–146.
Milner, R. (1993) Elements of Interaction. *Communications of the ACM* **36** 78–89.
Milner, R., Parrow, J. and Walker, D. (1992a) A Calculus of Mobile Processes: Part I. *Information and Computation* **100** 1–40.
Milner, R., Parrow, J. and Walker, D. (1992b) A Calculus of Mobile Processes: Part II. *Information and Computation* **100** 41–77.
Milner, R and Sangiorgi, D. (1992) Barbed Bisimulation. In: Proceedings ICALP'92. *Springer-Verlag 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. *Springer-Verlag 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 Cybernetics 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: *Proceedings 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. (2004a) Completeness Results for Undecidable Bisimilarity Problems. *Electronic Notes in Theoretical Computer Science* **98** 5–19.
Srba, J. (2004b) Roadmap of Infinite Results. In: *Formal Models and Semantics: II*, World Scientific.
Turing, A. (1936) On Computable Numbers, with an Application to the Entsheidungsproblem. *Proceedings of the London Mathematical Society* **42** 230–265.
Turing, A. (1937a) Computability and λ-Definability. *Journal of Symbolic Logic* **2** 153–163.
Turing, A. (1937b) 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. *Springer-Verlag 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.

https://doi.org/10.1017/S0960129514000012 Published online by Cambridge University Press