Sequential Composition in the Presence of Intermediate Termination

Jos Baeten
CWI
Amsterdam, the Netherlands
University of Amsterdam,
Amsterdam, the Netherlands
Jos.Baeten@cwi.nl

Bas Luttik
Eindhoven University of Technology
Eindhoven, The Netherlands
s.p.luttik@tue.nl

Fei Yang
Eindhoven University of Technology
Eindhoven, The Netherlands
f.yang@tue.nl

The standard operational semantics of the sequential composition operator gives rise to unbounded branching and forgetfulness when transparent process expressions are put in sequence. Due to transparency, the correspondence between context-free and push-down processes fails modulo bisimilarity, and it is not clear how to specify an always terminating half counter. We propose a revised operational semantics for the sequential composition operator in the context of intermediate termination. With the revised operational semantics, we eliminate transparency. As a consequence, we establish a correspondence between context-free processes and pushdown processes. Moreover, we prove the reactive Turing powerfulness of TCP with iteration and nesting with the revised operational semantics for sequential composition.

1 Introduction

The integration of the concurrency theory and the classical theory of formal languages has been extensively studied in recent years [2]. A lot of notions in the classical theory have their counterparts in the concurrency theory [14]. However, we still cannot conclude a complete correspondence for all the notions. As far as we are concerned, a major obstacle is the phenomenon of transparency of the sequential composition operator in the presence of intermediate termination.

Sequential composition is a standard operator in many process calculi. The functionality of the sequential composition operator is to concatenate the behaviours of two systems. It has been widely used in many process calculi with the notation "$\cdot$". We illustrate its operational semantics by a process $P \cdot Q$ in TCP [1]. If the process P has a transition $P \xrightarrow{a} P'$ for some action label a, then the composition $P \cdot Q$ has the transition $P \cdot Q \xrightarrow{a} P' \cdot Q$. Termination is an important behaviour for models of computation [1]. Combining with sequential composition, we have two additional rules in the operational semantics. The first one states that $P \cdot Q$ terminates if both P and Q terminates; and the second one states that if P terminates, and there is a transition $Q \xrightarrow{a} Q'$ for some action label a, then we have the transition $P \cdot Q \xrightarrow{a} Q'$. Together with the first rules, we are able to characterise the behaviour of systems consisting of two concatenated parts. The system may make a transition of the first part. If the first part is terminating, then the system may choose to skip the first part and make a transition of the second part. If both parts are terminating, then the combined system is also terminating.

In this paper, we discuss a complication on the standard version of the operational semantics of the sequential composition operator. The complication is on the transparency caused by sequential composition operator and termination [2]. A process expression is called transparent if it is terminating. Actually, it is then transparent in a "sequential context". We have observed two disadvantages on transparency in the following research problems.

Submitted to:
EXPRESS/SOS 2017

The relationship between context-free processes and pushdown automaton has been extensively discussed in the literature [3]. It has been shown that every context-free process can be specified by a pushdown process modulo contra simulation. However, such result is no longer valid modulo rooted branching bisimulation, which is a finer behavioural equivalence. By stacking unboundedly many transparent terms with sequential composition, we would get a transition system with unboundedly branching on its behaviour. It was shown that such unboundedly branching behaviour cannot be specified by any pushdown process modulo rooted branching bisimulation [3]. In order to improve the result to a finer notion of behaviour equivalence, we need to eliminate the problem of unboundedly branching.

Another problem is that transparency makes a stack of transparent process expressions forgetful. A notion of reactive Turing machines (RTM) [4] was introduced as a model to integrate concurrency and computability. The transition systems associated with RTMs are called executable. We use the RTM as a criteria of absolute expressiveness of process calculi and interactive computation models [10, 11]. A process calculus is called reactively Turing powerful if every executable transition system can be specified. The process calculus TCP with iteration and nesting is Turing complete [5, 6]. Moreover, it follows from the result in [6] that it is reactively Turing powerful if termination is out of consideration. However, it is not clear to us how to reconstruct the proof of reactive Turing powerfulness if termination is considered. Due to the forgetfulness on the stacking of transparent process expressions, it is not clear to us how to define a counter that is always terminating which is crucial to establish the reactive Turing powerfulness.

In order to avoid the unwanted feature of unbounded branching and forgetfulness, we propose a revised operational semantics for the sequential composition operator. Intuitively, we disallow the transition from the second component of a sequential composition if the first component is able to still perform a transition. Thus, we avoid the problems mentioned above with the revised operator. We shall prove that every context-free process is bisimilar to a pushdown process and TCP with iteration and nesting is reactively Turing powerful modulo divergence-preserving branching bisimilarity without using recursive specification in the revised semantics.

The paper is structured as follows. We first introduce TCP with the standard version of sequential composition in Section 2. Next, we discuss the complications caused by transparency in Section 3. Then, in Section 4, we propose the revised operational semantics of the sequential composition operator, and show that rooted divergence-preserving branching bisimulation is a congruence. In Section 5, we revisit the problem on the relationship between context-free processes and pushdown automaton, and show that every context-free process is bisimilar to a pushdown process in our revised semantics. In Section 6, we prove that TCP with iteration and nesting is a reactively Turing powerful in the revised semantics. In Section 7, we draw some conclusions and propose some future work.

2 Preliminaries

We start with introducing a notion of labelled transition systems which is used as the standard mathematical representation of behaviour. We consider transition systems with a subset of states marked with final states. We let $\mathcal{A}$ be a set of *action symbols*, and we extend $\mathcal{A}$ with a special symbol $\tau \notin \mathcal{A}$, which intuitively denotes unobservable internal activity of the system. We shall abbreviate $\mathcal{A} \cup \{\tau\}$ by $\mathcal{A}_\tau$.

Definition 1. *An* $\mathcal{A}_\tau$-labelled transition system *is a tuple* $(\mathcal{S}, \longrightarrow, \uparrow, \downarrow)$*, where*

1. $\mathcal{S}$ *is a set of* states*,*
2. $\longrightarrow \subseteq \mathcal{S} \times \mathcal{A}_\tau \times \mathcal{S}$ *is an* $\mathcal{A}_\tau$*-labelled* transition relation*,*

3. $\uparrow \in \mathcal{S}$ *is the initial state, and*
4. $\downarrow \subseteq \mathcal{S}$ *is a set of terminating states.*

Next, we shall use the process calculus TCP that allows us to describe transition systems. Let C be a set of *channels* and $\mathcal{D}_\Box$ be a set of *data symbols*. For every subset $C' \subseteq C$, we define a special set of actions $\mathcal{I}_{C'} \subseteq \mathcal{A}_\tau$ by:

$$\mathcal{I}_{C'} = \{c?d, c!d \mid d \in \mathcal{D}_\Box, c \in C'\} \ .$$

The actions $c?d$ and $c!d$ denote the events that a datum d is received or sent along channel c. Furthermore, let $\mathcal{N}$ be a countably infinite set of names. The set of process expressions $\mathcal{P}$ is generated by the following grammar ($a \in \mathcal{A}_\tau$, $N \in \mathcal{N}$, $C \in C$):

$$P = \mathbf{0} \mid \mathbf{1} \mid a.P \mid P_1 \cdot P_2 \mid [P_1 \parallel P_2]_{C'} \mid P_1 + P_2 \mid N \ .$$

We briefly comment on the operators in this syntax. The constant $\mathbf{0}$ denotes *deadlock*, the unsuccessfully terminated process. The constant $\mathbf{1}$ denotes *termination*, the successfully terminated process. For each action $a \in \mathcal{A}_\tau$ there is a unary operator $a.$ denoting action prefix; the process denoted by $a.P$ can do an a-labelled transition to the process P. The binary operator + denotes alternative composition or choice. The binary operator $[\_ \parallel \_]_C$ deviates from TCP in [1] which denotes a special kind of parallel composition. It enforces communication along the channels in C, and communication results in τ. The binary operator $\cdot$ represents the sequential composition of two processes.

Let P be an arbitrary process expression; and we use an abbreviation inductively defined by:

1. $P^0 = \mathbf{1}$; and
2. $P^{n+1} = P \cdot P^n$ for all $n \in \mathbb{N}$.

A recursive specification E is a set of equations

$$E = \{N \stackrel{\text{def}}{=} P | N \in \mathcal{N}, P \in \mathcal{P}\} \ .$$

satisfying the requirements that

1. for every $N \in \mathcal{N}$ it includes at most one equation with N as left-hand side, which is referred to as the *defining equation* for N; and
2. if some name N' occurs in the right-hand side P of some equation $N = p$ in E, then E must include a defining equation for N'.

A recursive specification is *guarded* if every summand in the specification that is not $\mathbf{1}$ is in the scope of some action prefix.

We use structural operational semantics to associate a transition relation with process expressions defined in TCP. We let $\longrightarrow$ be the $\mathcal{A}_\tau$-labelled transition relation induced on the set of process expressions by operational rules in Figure 1. Note that we presuppose a recursive specification E.

Here we use $P \xrightarrow{a} P'$ to denote an a-labelled transition $(P, a, P') \in \longrightarrow$. We say a process expression P' is *reachable* from P is there exists process expressions $P_0, P_1, \ldots, P_n$ and labels $a_1, \ldots, a_n$, such that $P = P_0 \xrightarrow{a_1} P_1 \ldots \xrightarrow{a_n} P_n = P'$.

Given a TCP process expression P, the transition system $\mathcal{T}(P) = (\mathcal{S}_P, \longrightarrow_P, \uparrow_P, \downarrow_P)$ associated with P is defined as follows:

1. the set of states $\mathcal{S}_P$ consists of all process expressions reachable from P;

$$\frac{}{\mathbf{1}\downarrow} \qquad \frac{}{a.P \xrightarrow{a} P}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 + P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 + P_2 \xrightarrow{a} P_2'} \qquad \frac{P_1\downarrow}{P_1 + P_2\downarrow} \qquad \frac{P_2\downarrow}{P_1 + P_2\downarrow}$$

$$\frac{P_1 \xrightarrow{a} P_1' \quad a \notin \mathcal{I}_{C'}}{[P_1 \parallel P_2]_{C'} \xrightarrow{a} [P_1' \parallel P_2]_{C'}} \qquad \frac{P_2 \xrightarrow{a} P_2' \quad a \notin \mathcal{I}_{C'}}{[P_1 \parallel P_2]_{C'} \xrightarrow{a} [P_1 \parallel P_2']_{C'}}$$

$$\frac{P_1 \xrightarrow{c?d} P_1' \quad P_2 \xrightarrow{c!d} P_2' \quad c \in C'}{[P_1 \parallel P_2]_{C'} \xrightarrow{\tau} [P_1' \parallel P_2']_{C'}} \qquad \frac{P_1 \xrightarrow{c!d} P_1' \quad P_2 \xrightarrow{c?d} P_2' \quad c \in C'}{[P_1 \parallel P_2]_{C'} \xrightarrow{\tau} [P_1' \parallel P_2']_{C'}} \qquad \frac{P_1\downarrow \quad P_2\downarrow}{[P_1 \parallel P_2]_{C'}\downarrow}$$

$$\frac{P_1\downarrow \quad P_2\downarrow}{P_1 \cdot P_2\downarrow} \qquad \frac{P_1 \xrightarrow{a} P_1'}{P_1 \cdot P_2 \xrightarrow{a} P_1' \cdot P_2} \qquad \frac{P_1\downarrow \quad P_2 \xrightarrow{a} P_2'}{P_1 \cdot P_2 \xrightarrow{a} P_2'}$$

$$\frac{P \xrightarrow{a} P' \quad (N \stackrel{\text{def}}{=} P) \in E}{N \xrightarrow{a} P'} \qquad \frac{P\downarrow \quad (N \stackrel{\text{def}}{=} P) \in E}{N\downarrow}$$

Figure 1: The operational Semantics of TCP

2. the set of transitions $\longrightarrow_P$ is the restriction to $\mathcal{S}_P$ of the transition relation defined on all process expressions by the structural operational semantics, i.e., $\mathcal{S}_P$ of the $\longrightarrow_P = \longrightarrow \cap (\mathcal{S}_P \times \mathcal{A}_\tau \times \mathcal{S}_P)$;
3. $\uparrow_P = P$; and
4. the set of final states $\downarrow_P$ consists of all process expressions $Q \in \mathcal{S}_P$ such that $Q\downarrow$, i.e., $\downarrow_P = \downarrow \cap \mathcal{S}_P$.

We also use the process calculus TSP in later sections. It is obtained by excluding the parallel composition operator from TCP.

The notion of behavioural equivalence has been used extensively in the theory of process calculi. We first introduce the notion of strong bisimulation [12, 13], which does not distinguish τ-transitions from other labelled transitions.

Definition 2. *A binary symmetric relation $\mathcal{R}$ on a transition system $(\mathcal{S}, \longrightarrow, \uparrow, \downarrow)$ is a* strong bisimulation *if, for all states $s, t \in \mathcal{S}$, $s\mathcal{R}t$ implies*

1. *if $s \xrightarrow{a} s'$, then there exist $t' \in \mathcal{S}_2$, such that $t \xrightarrow{a} t'$, and $s'\mathcal{R}t'$;*
2. *if $s\downarrow$, then $t\downarrow$.*

The states s and t are strongly bisimilar *(notation: $s \leftrightarroweq t$) if there exists a strong bisimulation $\mathcal{R}$ s.t. $s\mathcal{R}t$.*

The notion of strong bisimilarity does not take into account the intuition associated with τ that it stands for unobservable internal activity. To this end, we proceed to introduce the notion of (divergence-preserving) branching bisimilarity, which does treat τ-transitions as unobservable. Divergence-preserving branching bisimilarity in this paper which is the finest behavioural equivalence in van Glabbeek's linear time - branching time spectrum [8]. Let $\longrightarrow$ be an $\mathcal{A}_\tau$-labelled transition relation on a set $\mathcal{S}$, and let $a \in \mathcal{A}_\tau$; we write $s \xrightarrow{(a)} t$ for the formular "$s \xrightarrow{a} t \vee (a = \tau \wedge s = t)$". Furthermore, we denote the transitive closure of $\xrightarrow{\tau}$ by $\longrightarrow^+$ and the reflexive-transitive closure of $\xrightarrow{\tau}$ by $\longrightarrow^*$.

Definition 3. *Let $T = (\mathcal{S}, \longrightarrow, \uparrow, \downarrow)$ be a transition system. A branching bisimulation is a symmetric relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ such that for all states $s, t \in \mathcal{S}$, $s\mathcal{R}t$ implies*

1. *if $s \xrightarrow{a} s'$, then there exist $t', t'' \in \mathcal{S}_2$, such that $t \longrightarrow^* t'' \xrightarrow{(a)} t'$, $s\mathcal{R}t''$ and $s'\mathcal{R}t'$;*
2. *if $s\downarrow$, then there exists t' such that $t \longrightarrow^* t'$ and $t'\downarrow$; and*

The states s and t are branching bisimilar *(notation: $s \leftrightarrows_b t$) if there exists a branching bisimulation $\mathcal{R}$, s.t. $s\mathcal{R}t$.*

A branching bisimulation $\mathcal{R}$ is divergence-preserving *if, for all states s and t, $s\mathcal{R}t$ implies*

3. *if there exists an infinite sequence $(s_i)_{i\in\mathbb{N}}$ such that $s = s_0$, $s_i \xrightarrow{\tau} s_{i+1}$ and $s_i\mathcal{R}t$ for all $i \in \mathbb{N}$, then there exists a state t' such that $t \longrightarrow^+ t'$ and $s_i\mathcal{R}t'$ for some $i \in \mathbb{N}$.*

The states s and t are divergence-preserving branching bisimilar *(notation: $s \leftrightarrows^{\Delta}_b t$) if there exists a divergence-preserving branching bisimulation $\mathcal{R}$ such that $s\mathcal{R}t$.*

We call the largest divergence-preserving branching bisimulation relation *divergence-preserving branching bisimilarity*. Note that divergence-preserving branching bisimulation relations are equivalence relations [9].

Definition 4. *An equivalence relation $\mathcal{R}$ on a process calculus C is called a congruence if $s_i\mathcal{R}t_i$ for $i = 1, ..., ar(f)$ implies $f(s_1, \ldots, s_{ar(f)})\mathcal{R}f(t_1, \ldots, t_{ar(f)})$, where f is an operator of C, $ar(f)$ is the arity of f, and s_i, t_i are processes defined in C.*

Divergence-preserving branching bisimulation relation is not a congruence with respect to most process calculi. A rootedness condition needs to be introduced.

Definition 5. *A divergence-preserving branching bismulation relation $\mathcal{R}$ on a transition system $(\mathcal{S}, \longrightarrow, \uparrow, \downarrow)$ satisfies* rootedness *condition on a pair of states $s_1, s_2 \in \mathcal{S}$, if $s_1\mathcal{R}s_2$ and*

1. *if $s_1 \xrightarrow{a} s_1'$, then $s_2 \xrightarrow{a} s_2'$ for some s_2' such that $s_1'\mathcal{R}s_2'$;*
2. *if $s_2\downarrow$, then $s_1\downarrow$.*

s_1 and s_2 are rooted divergence-preserving branching bisimilar *(notation: $s_1 \leftrightarrows^{\Delta}_{rb} s_2$) if there exists a divergence-preserving branching bisimulation $\mathcal{R}$, such that $s_1\mathcal{R}s_2$, and it satisfies rootedness condition on s_1 and s_2.*

We can extend the above relations ($\leftrightarrows$, $\leftrightarrows_{\mathrm{b}}$, $\leftrightarrows^{\Delta}_{\mathrm{b}}$, and $\leftrightarrows^{\Delta}_{\mathrm{rb}}$) to relations over two transition systems by taking the union of two disjoint transition systems, and two transition systems are bisimilar if their initial states are bisimilar in the union. Namely, for two transition systems $T_1 = (\mathcal{S}_1, \longrightarrow_1, \uparrow_1, \downarrow_1)$ and $T_2 = (\mathcal{S}_2, \longrightarrow_2, \uparrow_2, \downarrow_2)$, we make the following pairing on their states. We pair every state $s \in \mathcal{S}_1$ with 1 and every state $s \in \mathcal{S}_2$ with 2. We have $T_i' = (\mathcal{S}_i', \longrightarrow_i', \uparrow_i', \downarrow_i')$ for $i = 1, 2$ where $\mathcal{S}_i' = \{(s, i) \mid s \in \mathcal{S}_i\}$, $\longrightarrow_i' = \{((s,i), a, (t,i)) \mid (s,a,t) \in \longrightarrow_i\}$, $\uparrow_i' = (\uparrow_i, i)$, and $\downarrow_i' = \{(s,i) \mid s \in \downarrow_i\}$. We say $T_1 \equiv T_2$ if their exists a behaviour equivalence $\equiv$ on $T = (\mathcal{S}_1' \cup \mathcal{S}_2', \longrightarrow_1' \cup \longrightarrow_2', \uparrow_1', \downarrow_1' \cup \downarrow_2')$ such that $\uparrow_1' \equiv \uparrow_2'$.

3 Transparency

Process expressions that have the option to terminate are *transparent* in a sequential context: if P has the option to terminate and $Q \xrightarrow{a} Q'$, then $P \cdot Q \xrightarrow{a} Q'$ even if P can still do transitions. In this section we shall explain how transparency gives rise to two phenomena that are undesirable in certain circumstances.

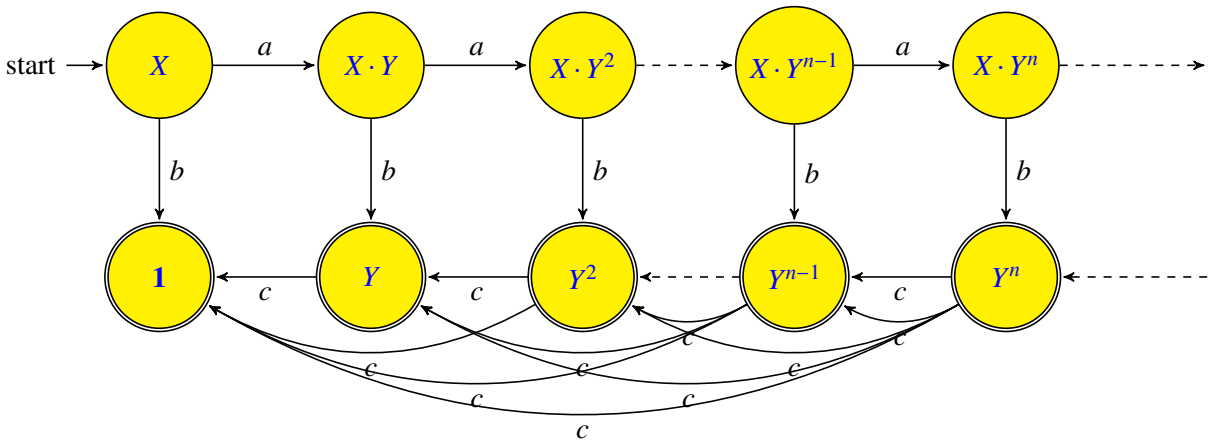

Figure 2: A transition system with unboundedly branching behaviour

First, it facilitates the specification of unboundedly branching behaviour with a guarded recursive specification over TSP. Second, it gives rise to forgetful stacking of variables, and as a consequence it is not clear how to specify an always terminating half-counter.

We first discuss process expressions with unbounded branching. I has been a well-known result from formal language theory that the context-free languages are exactly the languages accepted by pushdown automata. The process-theoretic formulation of this result is that every transition system specified by a TSP specification is language equivalent to the transition system associated with a pushdown automaton and, vice versa, every transition system associated with a pushdown automaton is language equivalent to the transition system associated with some TSP specification. The correspondence fails, however, when language equivalence is replaced by (strong) bisimilarity. The current best known result is that for every context-free process, there is a pushdown process to simulate it modulo contra simulation [3]. However, we have not succeeded in improving the result to branching bisimulation. The reason is that the context-free processes might have unbounded branching degree when there are unboundedly many transparency process expressions connected by sequential composition. Consider the following process:

$$X = a.X \cdot Y + b.\mathbf{1} \quad Y = c.\mathbf{1} + \mathbf{1} \ .$$

The transition system of the above process is illustrated in Figure 2. Note that X is the initial state, and every state in the second row is a terminating state. For the state Y^n, it has n c-labelled transitions to $\mathbf{1}, Y, Y^2, \ldots, Y^{n-1}$, respectively. Therefore, every state in this transition system has finitely many transitions leading to distinct states, whereas there is no upper bound on the number of transitions from each state. In this case, we say that this transition system has unboundedly branching degree.

we can prove that the process defined by the TSP specification above is not strongly bisimilar to a pushdown process since it is unbounded branching, whereas a pushdown process is always boundedly branching. This correspondence does hold for contra simulation [3], and that it is an open problem as to whether the correspondence holds modulo branching bisimilarity. In Section 5, we show that with the revised composition operator, we can remove such unbounded branching and establish a correspondence between pushdown process and context-free process modulo strong bisimilarity.

Now we proceed to discuss the phenomenon of forgetfulness. A process calculus with iteration and nesting is introduced by Bergstra, Bethke and Ponse [5, 6] in which a binary nesting operator $\sharp$ and a

$$\frac{}{P^* \downarrow} \qquad \frac{P \xrightarrow{a} P'}{P^* \xrightarrow{a} P' \cdot P^*}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \sharp P_2 \xrightarrow{a} P_1' \cdot P_1 \sharp P_2 \cdot P_1} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \sharp P_2 \xrightarrow{a} P_2'} \qquad \frac{P_2 \downarrow}{P_1 \sharp P_2 \downarrow} \ .$$

Figure 3: The operational semantics of nesting and iteration

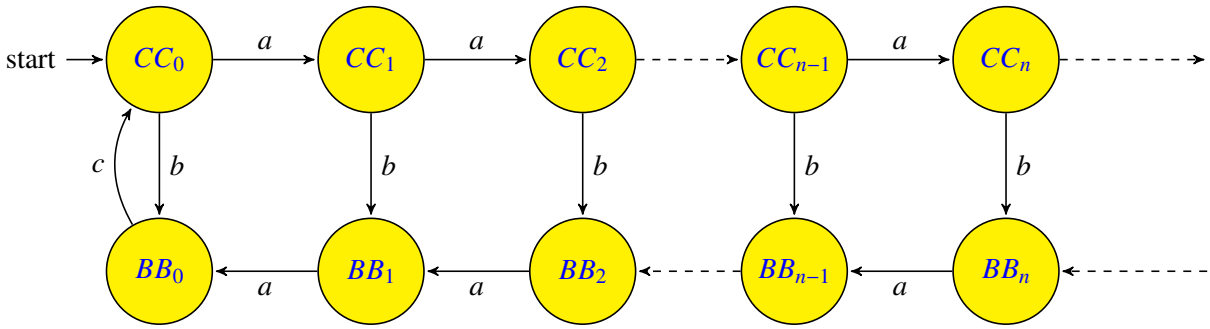

Figure 4: The transition system of a half counter

Kleene star operator * are added. In this paper, we add these two operators to TCP. They are intuitively given by the following equations where we use equality to represent strong bisimilarity:

$$P^* = P \cdot P^* + \mathbf{1} \quad P_1 \sharp P_2 = P_1 \cdot (P_1 \sharp P_1) \cdot P_1 + P_2$$

We give the operational semantics of these two operators in Figure 3.

Bergstra et al. show how one can specify a half counter using iteration and nesting, which then allows them to conclude that the behaviour of a Turing machine can be simulated in the calculus with iteration and nesting [5, 6].

The half counter is specified as follows:

$$\begin{aligned} CC_n &= a.CC_{n+1} + b.BB_n (n \in \mathbb{N}) \\ BB_n &= a.BB_{n-1} (n \geq 1) \\ BB_0 &= c.CC_0 \ . \end{aligned}$$

The behaviour of a half counter is illustrated in Figure 4. The initial state is CC_0. From CC_0, it is able to make an arbitrary number of a transitions. At some point, it stops counting with a b-labelled transition, and then makes the same number of a-labelled transitions to the state BB_0. In state BB_0, a zero testing transition is labelled by c which leads back to the state CC_0.

An implementation in TCP with iteration and nesting is provided in [6] as follows:

$$HCC = ((a \sharp b) \cdot c)^* \ .$$

It is straightforward to establish that $((a \sharp b) \cdot a^n \cdot c) \cdot HCC$ is equivalent to CC_n for all $n \geq 1$ modulo strong bisimilarity, and $(a^n \cdot c) \cdot HCC$ is equivalent to BB_n for all $n \in \mathbb{N}$ modulo strong bisimilarity.

$$\frac{P_1\downarrow \quad P_2\downarrow}{P_1;P_2\downarrow} \qquad \frac{P_1 \xrightarrow{a} P_1'}{P_1;P_2 \xrightarrow{a} P_1';P_2} \qquad \frac{P_1\downarrow \quad P_2 \xrightarrow{a} P_2' \quad P_1 \not\longrightarrow}{P_1;P_2 \xrightarrow{a} P_2'} \ .$$

Figure 5: The revised semantics of sequential composition

In a context with intermediate termination, one may wonder if it is possible to generalize their result. It is, however, not clear how to specify an always terminating half counter. At least, a naive generalisation of the specification of Bergstra et al. does not do the job. The culprit is forgetfulness. We define a half counter that terminates in every state as follows:

$$\begin{aligned} C_n &= a.C_{n+1} + b.B_n + \mathbf{1} \ (n \in \mathbb{N}) \\ B_n &= a.B_{n-1} + \mathbf{1} \ (n \geq 1) \\ B_0 &= c.C_0 + \mathbf{1} \ . \end{aligned}$$

The following implementation is no longer valid:

$$HC = ((a+\mathbf{1})^{\sharp}(b+\mathbf{1})\cdot(c+\mathbf{1}))^* \ .$$

Note that due to transparency, $((a+\mathbf{1})^n\cdot(c+\mathbf{1}))\cdot HC$ is no longer equivalent to B_n modulo any behavioural equivalence for $n > 1$ since B_n only has an a-labelled transition to B_{n-1} whereas the other process has at least $n+1$ transitions leading to $HC, (c+\mathbf{1})\cdot HC, (a+\mathbf{1})\cdot(c+\mathbf{1})\cdot HC, \ldots, (a+\mathbf{1})^{n-1}\cdot(c+\mathbf{1})\cdot HC$, respectively. This process may choose to "forget" the transparent process expressions that has been stacked with sequential composition operator. The forgetfulness leads to the failure to implement a terminating half counter.

In Section 6, we show that with the revised semantics, the forgetfulness would be eliminated. Therefore, we show that we are able to implement a terminating half counter with the revised semantics and we shall provide a reactively Turing powerful process calculus.

4 A Revised Semantics of the Sequential Composition Operator

We propose a calculus TCP$^;$ with a new sequential composition operator. Its syntax obtained by replacing the sequential composition operator $\cdot$ by ; in the syntax of TCP. Note that we also use the abbreviation of P^n as we did for the standard version of the sequential composition operator.

The structural operational semantics of ; is defined in Figure 5. We use $P \not\longrightarrow$ to denote that there does not exist a closed term P' such that $P \xrightarrow{a} P'$ is derivable from the operational rules. With the revised semantics, processes with intermediate termination (option to terminate and option to do an action) lose their transparency in a sequential context. As a consequence, the branching degree of a context-free process is bounded and sequential compositions may have the option to terminate, without being forgetful.

Let us revisit the example in Section 3. We rewrite it with the revised sequential composition operator:

$$\begin{aligned} X &= a.X;Y + b.\mathbf{1} \\ Y &= c.\mathbf{1} + \mathbf{1} \ . \end{aligned}$$

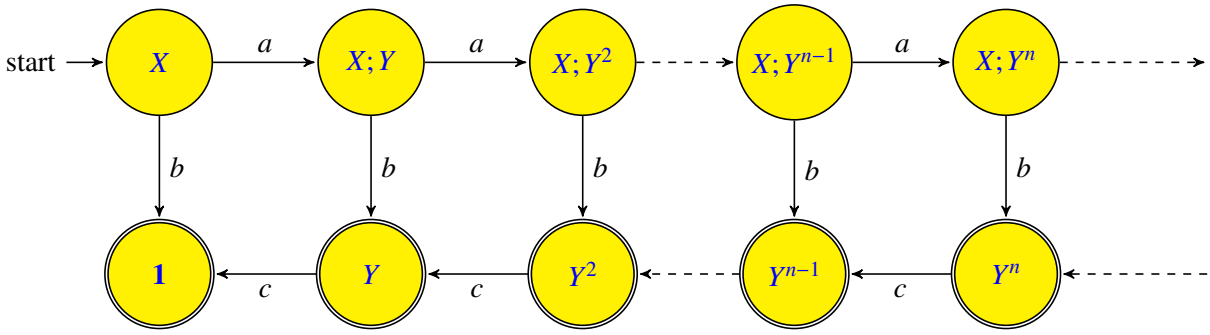

Figure 6: The transition system in the revised semantics

Its transition system is illustrated in Figure 6. Every state in the transition system now has a bounded branching degree. For instance, a transition from Y^5 to Y^2 is abandoned because Y has a transition and only the transition from the first Y in the sequential composition is allowed.

Congruence is an important property to fit a behavioural equivalence into an axiomatic framework. We show that in the revised semantics, $\leftrightarrows^{\Delta}_{\mathrm{rb}}$ is a congruence. Note that the congruence property can also be inferred from a recent result of Fokknik, van Glabbeek and Luttik [7].

Theorem 1. *$\leftrightarrows^{\Delta}_{rb}$ is a congruence with respect to TCP:.*

Proof. We use the following facts:

1. Rooted divergence-preserving branching bisimilarity is a rooted divergence-preserving branching bisimulation relation; and
2. rooted divergence-preserving branching bisimilarity is a subset of divergence-preserving branching bisimilarity.

We show that $\leftrightarrows^{\Delta}_{\mathrm{rb}}$ is compatible for each operator $a.,+,;,\|$.

1. Suppose that $P \leftrightarrows^{\Delta}_{\mathrm{rb}} Q$, we show that $a.P \leftrightarrows^{\Delta}_{\mathrm{rb}} a.Q$. To this end, we verify that $\mathcal{R} = \{(a.P, a.Q) \mid P \leftrightarrows^{\Delta}_{\mathrm{rb}} Q\} \cup \leftrightarrows^{\Delta}_{\mathrm{rb}}$ is a rooted divergence-preserving branching bisimulation relation.

   To prove that the pair $(a.P, a.Q)$ with P rooted divergence-preserving branching bisimilar to Q satisfies condition 1 of Definition , suppose that $a.P \xrightarrow{b} P'$. Then, according to the operational semantics, $b = a$ and $P' = P$. By the operational semantics, we also have that $a.Q \xrightarrow{a} Q$ and, by assumption, P and Q are divergence-preserving branching bisimilar.

   For the termination condition, it is trivially satisfied since both processes do not terminate. The divergence-preserving condition is also satisfied since only an a-labelled transition is allowed from both processes.

2. Suppose that $P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2$, we show that $P_1 + P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1 + Q_2$. To this end, we verify that $\mathcal{R} = \{(P_1 + P_2, Q_1 + Q_2) \mid P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1, P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2\} \cup \leftrightarrows^{\Delta}_{\mathrm{rb}}$ is a rooted divergence-preserving branching bisimulation relation.

   Suppose that $P_1 + P_2 \xrightarrow{a} P'$; then we have $P_1 \xrightarrow{a} P'$ or $P_2 \xrightarrow{a} P'$. We only consider the first case. Since $P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1$, we have $Q_1 \xrightarrow{a} Q'$ with $P' \leftrightarrows^{\Delta}_{\mathrm{b}} Q'$. Then we have $Q_1 + Q_2 \xrightarrow{a} Q'$ with $P' \leftrightarrows^{\Delta}_{\mathrm{b}} Q'$. The same argument holds for the symmetrical case.

   If $P_1 + P_2 \downarrow$, then we have either $P_1 \downarrow$ or $P_2 \downarrow$. Without loss of generality, we suppose that $P_1 \downarrow$. Since $P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1$, we have $Q_1 \downarrow$. Therefore, $Q_1 + Q_2 \downarrow$.

Moreover, we verify that the divergence preservation condition is satisfied.

Hence, $\mathcal{R}$ is a rooted divergence-preserving branching bisimulation relation.

3. Suppose that $P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2$, we show that $[P_1 \parallel P_2]_{C'} \leftrightarrows^{\Delta}_{\mathrm{rb}} [Q_1 \parallel Q_2]_{C'}$. To this end, we verify that $\mathcal{R} = \{([P_1 \parallel P_2]_{C'}, [Q_1 \parallel Q_2]_{C'}) \mid P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1, P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2\} \cup \leftrightarrows^{\Delta}_{\mathrm{rb}}$ is a rooted divergence-preserving branching bisimulation relation.

   We first show that $\mathcal{R}' = \{([P_1 \parallel P_2]_{C'}, [Q_1 \parallel Q_2]_{C'}) \mid P_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_1, P_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_2\} \cup \leftrightarrows^{\Delta}_{\mathrm{b}}$ is a divergence-preserving branching bisimulation.

   Suppose that $[P_1 \parallel P_2]_{C'} \xrightarrow{a} P'$; then we distinguish several cases.

   (a) If $P_1 \xrightarrow{a} P'_1$ $a \notin \mathcal{I}_{C'}$ and $P' = [P'_1 \parallel P_2]_{C'}$, then, since $P_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_1$, we have $Q_1 \longrightarrow^* Q''_1 \xrightarrow{a} Q'_1$ with $P'_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_1$ and $P_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q''_1$. Then we have $[Q_1 \parallel Q_2]_{C'} \longrightarrow^* [Q''_1 \parallel Q_2]_{C'} \xrightarrow{a} [Q'_1 \parallel Q_2]_{C'}$ with $P_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q''_1$, $P'_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_2$. Thus we have $([P'_1 \parallel P_2]_{C'}, [Q'_1 \parallel Q_2]_{C'}) \in \mathcal{R}'$ and $([P_1 \parallel P_2]_{C'}, [Q''_1 \parallel Q_2]_{C'}) \in \mathcal{R}'$.

   (b) If $P_1 \xrightarrow{c?d} P'_1$, $P_2 \xrightarrow{c!d} P'_2$ and $c \in C'$, then $[P_1 \parallel P_2]_{C'} \xrightarrow{\tau} [P'_1 \parallel P'_2]_{C'}$. Since $P_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_2$, we have $Q_1 \longrightarrow^* Q''_1 \xrightarrow{c?d} Q'_1$, $Q_2 \longrightarrow^* Q''_2 \xrightarrow{c!d} Q'_2$ with $P'_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_1$, $P \leftrightarrows^{\Delta}_{\mathrm{b}} Q''_1$, $P'_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_2$, and $P_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q''_2$. Then we have $[Q_1 \parallel Q_2]_{C'} \longrightarrow^* [Q''_1 \parallel Q''_2]_{C'} \xrightarrow{\tau} [Q'_1 \parallel Q'_2]_{C'}$ with $P_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q''_1$, $P_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q''_2$, $P'_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_1$ and $P'_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_2$. Thus we have $([P_1 \parallel P_2]_{C'}, [Q''_1 \parallel Q''_2]_{C'}) \in \mathcal{R}'$ and $([P'_1 \parallel P'_2]_{C'}, [Q'_1 \parallel Q'_2]_{C'}) \in \mathcal{R}'$.

   If $[P_1 \parallel P_2]_{C'} \downarrow$, then we have $P_1 \downarrow$ and $P_2 \downarrow$. Since $P_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_2$, we have $Q_1 \longrightarrow^* Q'_1 \downarrow$ and $Q_2 \longrightarrow^* Q'_2 \downarrow$ for some Q'_1 and Q'_2. Therefore, $[Q_1 \parallel Q_2]_{C'} \longrightarrow^* [Q'_1 \parallel Q'_2]_{C'} \downarrow$.

   Hence, we have $\mathcal{R}'$ is a divergence-preserving branching bisimulation relation.

   Now we show that $\mathcal{R}$ is a rooted divergence-preserving branching bisimulation. Suppose that $[P_1 \parallel P_2]_{C'} \xrightarrow{a} P'$; then we distinguish several cases.

   (a) If $P_1 \xrightarrow{a} P'_1$ $a \notin \mathcal{I}_{C'}$ and $P' = [P'_1 \parallel P_2]_{C'}$, then, since $P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1$, we have $Q_1 \xrightarrow{a} Q'_1$ with $P'_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_1$. Then we have $[Q_1 \parallel Q_2]_{C'} \xrightarrow{a} [Q'_1 \parallel Q_2]_{C'}$ with $P'_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_2$. Thus we have $[P'_1 \parallel P_2]_{C'} \leftrightarrows^{\Delta}_{\mathrm{b}} [Q'_1 \parallel Q_2]_{C'}$.

   (b) If $P_1 \xrightarrow{c?d} P'_1$, $P_2 \xrightarrow{c!d} P'_2$ and $c \in C'$, then $[P_1 \parallel P_2]_{C'} \xrightarrow{\tau} [P'_1 \parallel P'_2]_{C'}$. Since $P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2$, we have $Q_1 \xrightarrow{c?d} Q'_1$, $Q_2 \xrightarrow{c!d} Q'_2$ with $P'_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_1$ and $P'_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_2$. Then we have $[Q_1 \parallel Q_2]_{C'} \xrightarrow{\tau} [Q'_1 \parallel Q'_2]_{C'}$ with $P'_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_1$ and $P'_2 \leftrightarrows^{\Delta}_{\mathrm{b}} Q'_2$. Thus we have $[P'_1 \parallel P'_2]_{C'} \leftrightarrows^{\Delta}_{\mathrm{b}} [Q'_1 \parallel Q'_2]_{C'}$.

   If $[P_1 \parallel P_2]_{C'} \downarrow$, then we have $P_1 \downarrow$ and $P_2 \downarrow$. Since $P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2$, we have $Q_1 \downarrow$ and $Q_2 \downarrow$. Therefore, $[Q_1 \parallel Q_2]_{C'} \downarrow$.

   Moreover, we verify that the divergence preservation condition is satisfied.

   Hence, we have $\mathcal{R}$ is a rooted divergence-preserving branching bisimulation relation.

4. Suppose that $P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1$ and $P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2$, we show that $P_1; P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1; Q_2$. To this end, we verify that $\mathcal{R} = \{(P_1; P_2, Q_1; Q_2) \mid P_1 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_1, P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2\} \cup \leftrightarrows^{\Delta}_{\mathrm{rb}}$ is a rooted divergence-preserving branching bisimulation relation.

   We first show that $\mathcal{R}' = \{(P_1; P_2, Q_1; Q_2) \mid P_1 \leftrightarrows^{\Delta}_{\mathrm{b}} Q_1, P_2 \leftrightarrows^{\Delta}_{\mathrm{rb}} Q_2\} \cup \leftrightarrows^{\Delta}_{\mathrm{b}}$ is a divergence-preserving branching bisimulation relation.

   Suppose that $P_1; P_2 \xrightarrow{a} P'$; then we distinguish several cases.

$$\frac{}{P^* \downarrow} \qquad \frac{P \xrightarrow{a} P'}{P^* \xrightarrow{a} P'; P^*}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \sharp P_2 \xrightarrow{a} P_1'; P_1 \sharp P_2; P_1} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \sharp P_2 \xrightarrow{a} P_2'} \qquad \frac{P_2 \downarrow}{P_1 \sharp P_2 \downarrow}$$

Figure 7: The revised semantics of iteration and nesting

(a) If $P_1 \xrightarrow{a} P_1'$, then $P' = P_1'; P_2$. Since $P_1 \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1$, we have $Q_1 \longrightarrow^* Q_1'' \xrightarrow{a} Q_1'$ with $P_1' \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1'$ and $P_1 \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1''$. Then we have $Q_1; Q_2 \longrightarrow^* Q_1''; Q_2 \xrightarrow{a} Q_1'; Q_2$ with $P_1 \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1''$, $P_1' \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1'$, and $P_2 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_2$. Thus, we have $(P_1'; P_2, Q_1'; Q_2) \in \mathcal{R}'$ and $(P_1; P_2, Q_1''; Q_2) \in \mathcal{R}'$.

(b) If $P_1 \downarrow, P_2 \xrightarrow{a} P_2'$ and $P_1 \not\longrightarrow$. Since $P_1 \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1$ and $P_2 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_2$, we have $Q_1 \longrightarrow^* Q_1' \downarrow$, $Q_1' \not\longrightarrow$ for some Q_1' with $P_1 \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1'$, and $Q_2 \xrightarrow{a} Q_2'$, with $P_2' \leftrightarrows_{\mathrm{b}}^{\Delta} Q_2'$. Then, we have $Q_1; Q_2 \longrightarrow^* Q_1'; Q_2 \xrightarrow{a} Q_2'$ with $P_2' \leftrightarrows_{\mathrm{b}}^{\Delta} Q_2'$ and $P_1 \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1'$. Thus we have $(P_2', Q_2') \in \mathcal{R}'$ and $(P_1; P_2, Q_1'; Q_2) \in \mathcal{R}$.

If $P_1; P_2 \downarrow$, then we have $P_1 \downarrow$ and $P_2 \downarrow$. Since $P_1 \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1$ and $P_2 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_2$, we have $Q_1 \longrightarrow^* Q_1' \downarrow$ for some Q_1' and $Q_2 \downarrow$. Therefore, $Q_1; Q_2 \longrightarrow^* Q_1'; Q_2 \downarrow$.

Moreover, we verify that the divergence preservation condition is satisfied.

Hence, we have $\mathcal{R}$ is a divergence-preserving branching bisimulation relation.

Now we show that $\mathcal{R}$ is a rooted divergence-preserving branching bisimulation relation.

We suppose that $P_1; P_2 \xrightarrow{a} P'$, we distinguish several cases:

(a) If $P_1 \xrightarrow{a} P_1'$, then $P' = P_1'; P_2$. Since $P_1 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_1$, we have $Q_1 \xrightarrow{a} Q_1'$ with $P_1' \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1'$. Then we have $Q_1; Q_2 \xrightarrow{a} Q_1'; Q_2$ with $P_1' \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1'$ and $P_2 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_2$. Thus, we have $P_1'; P_2 \leftrightarrows_{\mathrm{b}}^{\Delta} Q_1'; Q_2$.

(b) If $P_1 \downarrow, P_2 \xrightarrow{a} P_2'$ and $P_1 \not\longrightarrow$. Since $P_1 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_1$ and $P_2 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_2$, we have $Q_1 \downarrow$, $Q_2 \xrightarrow{a} Q_2'$, with $P_2' \leftrightarrows_{\mathrm{b}}^{\Delta} Q_2'$, and $Q_1 \not\longrightarrow$. Then, we have $Q_1; Q_2 \xrightarrow{a} Q_2'$ with $P_2' \leftrightarrows_{\mathrm{b}}^{\Delta} Q_2'$.

If $P_1; P_2 \downarrow$, then we have $P_1 \downarrow$ and $P_2 \downarrow$. Since $P_1 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_1$ and $P_2 \leftrightarrows_{\mathrm{rb}}^{\Delta} Q_2$, we have $Q_1 \downarrow$ and $Q_2 \downarrow$. Therefore, $Q_1; Q_2 \downarrow$.

Moreover, we verify that the divergence preservation condition is satisfied.

Hence, we have $\mathcal{R}$ is a rooted divergence-preserving branching bisimulation relation.

□

We also define a version of TCP with iteration and nesting ($\mathrm{TCP}^{\sharp}$) in the revised semantics. By removing recursive specification and adding a non-regular iterator, we get $\mathrm{TCP}^{\sharp}$ as a variation of $\mathrm{TCP}^{;}$ with two additional operators: $P^*, P_1 \sharp P_2$. The operational semantics is defined in Figure 7.

5 Context-free Processes and Pushdown Processes

The relationship between context-free processes and pushdown processes has been studied extensively in the literature [3].

We consider the process calculus *Theory of Sequential Processes* (TSP$^:$) with the revised semantics of sequential composition operator as follows:

$$P = \mathbf{0} \mid \mathbf{1} \mid a.P \mid P_1; P_2 \mid P_1 + P_2 \mid N \ .$$

We give the definition of context-free processes as follows:

Definition 6. *A* context-free process *is the bisimulation equivalence class of the transition system generated by a finite guarded recursive specification over Sequential Algebra TSP$^:$.*

Note that every context-free process can be rewritten into a Greibach normal form. In this paper, we only consider all the context-free processes (guarded recursive specifications) written in Greibach normal form:

$$X = \sum_{i \in I_X} \alpha_i.\xi_i (+\mathbf{1}) \ .$$

In this form, every right-hand side of every equation consists of a number of summands, indexed by a finite set I_X (the empty sum is $\mathbf{0}$), each of which is $\mathbf{1}$, or of the form $\alpha_i.\xi_i$, where ξ_i is the sequential composition of a number of names (the empty sequence is $\mathbf{1}$).

We shall show that every context-free process is equivalent to a pushdown automata modulo strong bisimilarity. The notion of pushdown automata is defined as follows:

Definition 7. *A pushdown automaton (PDA) is a* 7*-tuple* $(\mathcal{S}, \Sigma, \mathcal{D}, \longrightarrow, \uparrow, Z, \downarrow)$*, where*

1. $\mathcal{S}$ *is a finite set of* states*,*
2. Σ *is a finite set of* input symbols*,*
3. $\mathcal{D}$ *is a finite set of* stack symbols*,*
4. $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{D} \times \Sigma^* \mathcal{S}$ *is a finite* transition relation*, (we write* $s \xrightarrow{a[d/\delta]} t$ *for* $(s, d, a, \delta, t) \in \longrightarrow$*),*
5. $\uparrow \in \mathcal{S}$ *is the* initial state*,*
6. $Z \in \mathcal{D}$ *is the* initial stack symbol*, and*
7. $\downarrow \subseteq \mathcal{S}$ *is a set of* accepting states*.*

We use a sequence of stack symbols $\delta \in \mathcal{D}^*$ to represent the contents of a stack. We associate with every pushdown automaton a labelled transition system. We call the transition system associated with a pushdown automaton a pushdown process.

Definition 8. *Let* $\mathcal{M} = (\mathcal{S}, \Sigma, \mathcal{D}, \longrightarrow, \uparrow, Z, \downarrow)$ *be a PDA. The* pushdown process $\mathcal{T}(\mathcal{M}) = (\mathcal{S}_\mathcal{T}, \longrightarrow_\mathcal{T}, \uparrow_\mathcal{T}, \downarrow_\mathcal{T})$ *associated* $\mathcal{M}$ *is defined as follows:*

1. *its set of states is the set* $\mathcal{S}_\mathcal{T} = \{(s, \delta) \mid s \in \mathcal{S}, \delta \in \mathcal{D}^*\}$ *of all configurations of* $\mathcal{M}$*;*
2. *its transition relation* $\longrightarrow_\mathcal{T} \subset \mathcal{S}_\mathcal{T} \times \mathcal{A}_\tau \times \mathcal{S}_\mathcal{T}$ *is the least relation satisfying, for all* $a \in \Sigma$*,* $d \in \mathcal{D}$*,* $\delta, \delta' \in \mathcal{D}^*$*:* $(s, d\delta) \xrightarrow{a}_\mathcal{T} (t, \delta'\delta)$ *iff* $s \xrightarrow{a[d/\delta']} t$*.*
3. *its initial state is the configuration* $\uparrow_\mathcal{T} = (\uparrow, Z)$*, and*
4. *its set of final states is the set* $\downarrow_\mathcal{T} = \{(s, \delta) \mid s \in \mathcal{S}, s\downarrow, \delta \in \mathcal{D}^*\}$*.*

Consider a context-free process in Greibach normal form defined by a set of names $\mathcal{V} = \{X_0, X_1, \ldots, X_m\}$ where

$$X_j = \sum_{i \in I_{X_j}} \alpha_{ij}.\xi_{ij}(+\mathbf{1}) \ ,$$

and X_0 is the initial state.

We introduce the following functions for a sequence of names ξ

1. $length : \mathcal{V}^* \to \mathbb{N}$, $length(\xi)$ computes the length of ξ;
2. $get : \mathcal{V}^* \times \mathbb{N} \to \mathcal{V}$, $get(\xi, i)$ computes the i-th name of ξ;
3. $suffset : \mathcal{V}^* \times \mathbb{N} \to 2^{|\mathcal{V}|}$, $suffset(\xi, i) = \{get(\xi, j) \mid j = i+1, \ldots length(\xi)\}$ computes the set that contains all the names in the suffix which starts from the i-th name of ξ.

Next we define a PDA $\mathcal{M} = (\mathcal{S}, \Sigma, \mathcal{D}, \longrightarrow, \uparrow, Z, \downarrow)$ to simulate the transition system associated with X_0 as follows:

1. $\mathcal{S} = \{s_D \mid D \subseteq \mathcal{V}\}$;
2. $\Sigma = \mathcal{A}_\tau$;
3. $\mathcal{D} = \mathcal{V} \cup \{X^\dagger \mid X \in \mathcal{V}\}$;
4. the set of transitions $\longrightarrow$ is defined as follows:

$$\begin{aligned} \longrightarrow \ &= \ \{(s_D, X_j^\dagger, \alpha_{ij}, \delta(s_D, X_j^\dagger, \xi_{ij}), s_{D(s_D, X_j^\dagger, \xi_{ij})}) \mid i \in I_{X_j},\ j = 1, \ldots, n,\ D \subset \mathcal{V}\} \\ &\cup \ \{(s_D, X_j, \alpha_{ij}, \delta(s_D, X_j, \xi_{ij}), s_{D(s_D, X_j, \xi_{ij})}) \mid i \in I_{X_j},\ j = 1, \ldots, n,\ D \subset \mathcal{V}\} \ . \end{aligned}$$

   where $\delta(s_D, X_j^\dagger, \xi_{ij})$ is a string of length $length(\xi_{ij})$ defined as follows: for $k = 1, \ldots, length(\xi_{ij})$, we let $X_k = get(\xi_{ij}, k)$,

   (a) if $X_k \notin (D/\{X_j\}) \cup suffset(\xi_{ij}, k)$, then the k-th symbol of $\delta(s_D, X_j^\dagger, \xi_{ij})$ is $X_k^\dagger$,

   (b) otherwise, the k-th symbol of $\delta(s_D, X_j^\dagger, \xi_{ij})$ is X_k,

   $\delta(s_D, X_j, \xi_{ij})$ is a string of length $length(\xi_{ij})$ defined as follows: for $k = 1, \ldots, length(\xi_{ij})$, we let $X_k = get(\xi_{ij}, k)$,

   (a) if $X_k \notin D \cup suffset(\xi_{ij}, k)$, then the k-th symbol of $\delta(s_D, X_j, \xi_{ij})$ is $X_k^\dagger$,

   (b) otherwise, the k-th symbol of $\delta(s_D, X_j, \xi_{ij})$ is X_k, and

   we also define $D(s_D, X_j^\dagger, \xi_{ij}) = (D/\{X_j\}) \cup suffset(\xi_{ij}, 0)$ and $D(s_D, X_j, \xi_{ij}) = D \cup suffset(\xi_{ij}, 0)$;
5. $\uparrow = s_{\{X_0\}}$;
6. $Z = X_0^\dagger$;
7. $\downarrow = \{s_D \mid \text{if for all } X \in D, X \downarrow\}$.

We observe that every process expression ξ is simulated by a configuration of $\mathcal{M}$ such that the sequence of names in ξ is stored in the stack. The first appearance of every name from the bottom of the stack is marked with †. The state is marked by a set that contains all the names in ξ. A state is terminating if and only if all the names in the set that marks the state are terminating. We show the following result:

Lemma 1. $\mathcal{T}(X_0) \leftrightarrows \mathcal{T}(\mathcal{M})$.

Proof. We first define an auxiliary function $stack : \mathcal{V}^* \to \mathcal{D}^*$ as follows: given $\xi \in \mathcal{V}^*$, for $k = 1, \ldots, length(\xi)$, we let $X_k = get(\xi, k)$,

1. if $X_k \notin suffset(\xi, k)$, then the k-th the element of $stack(\xi)$ is $X_k^\dagger$,
2. otherwise, the k-the the element of $stack(\xi)$ is X_k,

Note that $stack(X\xi)$ and $stack(\xi)$ share the same suffix of length $length(\xi)$.

We show that the following relation:

$$\mathcal{R} = \{(\xi, (s_{suffset(\xi,0)}, stack(\xi))) \mid \xi \in \mathcal{V}^*\} \ ,$$

is a strong bisimulation.

We rewrite ξ as $X_j\xi'$, then it has the following transitions:

$$X_j\xi' \xrightarrow{\alpha_{ij}} \xi_{ij}\xi', i \in I_{X_j} \ .$$

We need to show that they are simulated by the transitions:

$$(s_{suffset(\xi,0)}, stack(\xi)) \xrightarrow{\alpha_{ij}} (s_{suffset(\xi_{ij}\xi',0)}, stack(\xi_{ij}\xi')), i \in I_{X_j} \ .$$

Thus we have $(x_{ij}, (s_{suffset(\xi_{ij}\xi',0)}, stack(\xi_{ij}\xi'))) \in \mathcal{R}$.

We consider the configuration $(s_{suffset(\xi,0)}, stack(\xi))$, we distinguish two cases of the top symbol of the stack.

1. If $get(stack(\xi), 1) = X_j^\dagger$, then $\mathcal{M}$ has the transition

   $$(s_{suffset(\xi,0)}, X_j^\dagger, \alpha_{ij}, \delta(s_{suffset(\xi,0)}, X_j^\dagger, \xi_{ij}), s_{D(s_{suffset(\xi,0)}, X_j^\dagger, \xi_{ij})}) \ .$$

   The new stack is $S = \delta(s_{suffset(\xi,0)}, X_j^\dagger, \xi_{ij})stack(\xi')$. We verify that $S = stack(()\xi_{ij}\xi')$. Note that they share the same suffix $stack(\xi')$. We only needs to verify the first $length(\xi_{ij})$ elements. For the l-th element, we let $X_l = get(\xi ij, l)$, and we distinguish with two cases.

   (a) If $X_l \notin (suffset(\xi, 0)/\{X_j\}) \cup suffset(\xi_{ij}, l)$, then the l-th element of S is $X_l^\dagger$. Since $get(stack(\xi), 1) = X_j^\dagger$, from the definition of $stack$, we have $X_j \notin suffset(\xi, 1) = suffset(\xi', 0)$. Therefore, $suffset(\xi, 0)/\{X_j\} = suffset(\xi', 0)$. In this case, $X_l \notin suffset(\xi', 0) \cup suffset(\xi_{ij}, l)$. Moreover, we have $X_l \notin suffset(\xi_{ij}\xi', l)$, therefore, the l-th element of $stack(\xi_{ij}\xi')$ is also $X_l^\dagger$.

   (b) Otherwise, then the l-th element of S is X_l. By the definition of $stack$, we get that the l-th element of $stack(\xi_{ij}\xi')$ is also X_l.

   Moreover, we verify that the new state $s_{D(s_{suffset(\xi,0)}, X_j^\dagger, \xi_{ij})} = s_{suffset(\xi_{ij}\xi',0)}$. Note that we have

   $$\begin{aligned} &D(s_{suffset(\xi,0)}, X_j^\dagger, \xi_{ij}) = (suffset(\xi, 0)/\{X_j\}) \cup suffset(\xi_{ij}, 0) \\ &= suffset(\xi', 0) \cup suffset(\xi_{ij}, 0) = suffset(\xi_{ij}\xi', 0) \ . \end{aligned}$$

   Hence, we have $(s_{suffset(\xi,0)}, stack(\xi)) \xrightarrow{\alpha_{ij}} (s_{suffset(\xi_{ij}\xi',0)}, stack(\xi_{ij}\xi'))$.

2. if $get(stack(\xi), 1) = X_j$, then $\mathcal{M}$ has the transition

   $$(s_{suffset(\xi,0)}, X_j, \alpha_{ij}, \delta(s_{suffset(\xi,0)}, X_j, \xi_{ij}), s_{D(s_{suffset(\xi,0)}, X_j, \xi_{ij})}) \ ,$$

   The ne stack is $S = \delta(s_{suffset(\xi,0)}, X_j, \xi_{ij})stack(\xi')$. We verify that $S = stack(()\xi_{ij}\xi')$. Note that they share the same suffix $stack(\xi')$. We only needs to verify the first $length(\xi_{ij})$ elements. For the l-th element, we let $X_l = get(\xi ij, l)$, and we distinguish with two cases.

(a) If $X_l \notin (\mathit{suffset}(\xi,0)) \cup \mathit{suffset}(\xi_{ij},l)$, then the l-th element of S is $X_l^\dagger$. Since $\mathit{get}(\mathit{stack}(\xi),1) = X_j$, from the definition of *stack*, we have $X_j \in \mathit{suffset}(\xi,1) = \mathit{suffset}(\xi',0)$. Therefore, $\mathit{suffset}(\xi,0) = \mathit{suffset}(\xi',0)$. In this case, $X_l \notin \mathit{suffset}(\xi',0) \cup \mathit{suffset}(\xi_{ij},l)$. Moreover, we have $X_l \notin \mathit{suffset}(\xi_{ij}\xi',l)$, therefore, the l-th element of $\mathit{stack}(\xi_{ij}\xi')$ is also $X_l^\dagger$.

(b) Otherwise, then the l-th element of S is X_l. By the definition of *stack*, we get that the l-th element of $\mathit{stack}(\xi_{ij}\xi')$ is also X_l.

Moreover, we verify that the new state $s_{D(s_{\mathit{suffset}(\xi,0)},X_j,\xi_{ij})} = s_{\mathit{suffset}(\xi_{ij}\xi',0)}$. Note that we have

$$\begin{aligned} &D(s_{\mathit{suffset}(\xi,0)},X_j,\xi_{ij}) = \mathit{suffset}(\xi,0) \cup \mathit{suffset}(\xi_{ij},0) \\ &= \mathit{suffset}(\xi',0) \cup \mathit{suffset}(\xi_{ij},0) = \mathit{suffset}(\xi_{ij}\xi',0) \ . \end{aligned}$$

Hence, we have $(s_{\mathit{suffset}(\xi,0)}, \mathit{stack}(\xi)) \xrightarrow{\alpha_{ij}} (s_{\mathit{suffset}(\xi_{ij}\xi',0)}, \mathit{stack}(\xi_{ij}\xi'))$.

By concluding the two cases, the above transitions are correct.

Using a similar analysis, we also have all the transitions from $(s_{\mathit{suffset}(\xi,0)}, \mathit{stack}(\xi))$ are simulated by $X_j\xi'$.

Now we consider the termination condition. $\xi\downarrow$ iff for all $X \in \mathit{suffset}(\xi,0)$, $X\downarrow$. Note that $(s_{\mathit{suffset}(\xi,0)}, \mathit{stack}(\xi))\downarrow$ iff for all $X \in \mathit{suffset}(\xi,0)$, $X\downarrow$. Therefore, termination condition is also verified.

Hence, we have $\mathcal{T}(X_0) \leftrightarrows \mathcal{T}(\mathcal{M})$. □

We have the following theorem.

Theorem 2. *For every context-free process P, there exists a PDA $\mathcal{M}$, such that $\mathcal{T}(P) \leftrightarrows \mathcal{T}(\mathcal{M})$.*

6 Executability in the Context of Termination

In this section, we shall discuss the theory of executability in the context of termination. We shall prove that $\mathrm{TCP}^\sharp$ is reactively Turing powerful in the context of termination.

The notion of reactive Turing machines (RTM) [4] was introduced as an extension of Turing machines to define which behaviour is executable by a computing system in terms of labelled transition systems. The definition of RTMs is parameterised with the set $\mathcal{A}_\tau$, which we assume to be a finite set. Furthermore, the definition is parameterised with another finite set $\mathcal{D}$ of *data symbols*. We extend $\mathcal{D}$ with a special symbol $\Box \notin \mathcal{D}$ to denote a blank tape cell, and denote the set $\mathcal{D} \cup \{\Box\}$ of *tape symbols* by $\mathcal{D}_\Box$.

Definition 9 (Reactive Turing Machine)**.** *A* reactive Turing machine *(RTM) is a quadruple $(\mathcal{S}, \longrightarrow, \uparrow, \downarrow)$, where*

1. *$\mathcal{S}$ is a finite set of* states*,*
2. *$\longrightarrow \subseteq \mathcal{S} \times \mathcal{D}_\Box \times \mathcal{A}_\tau \times \mathcal{D}_\Box \times \{L,R\} \times \mathcal{S}$ is a finite collection of $(\mathcal{D}_\Box \times \mathcal{A}_\tau \times \mathcal{D}_\Box \times \{L,R\})$-labelled* transitions *(we write $s \xrightarrow{a[d/e]M} t$ for $(s,d,a,e,M,t) \in \longrightarrow$),*
3. *$\uparrow \in \mathcal{S}$ is a distinguished* initial state*.*
4. *$\downarrow \subseteq \mathcal{S}$ is a finite set of* final states*.*

Intuitively, the meaning of a transition $s \xrightarrow{a[d/e]M} t$ is that whenever the RTM is in state s, and d is the symbol currently read by the tape head, then it may execute the action a, write symbol e on the tape (replacing d), move the read/write head one position to the left or the right on the tape (depending on whether $M = L$ or $M = R$), and then end up in state t.

To formalise the intuitive understanding of the operational behaviour of RTMs, we associate with every RTM $\mathcal{M}$ an $\mathcal{A}_\tau$-labelled transition system $\mathcal{T}(\mathcal{M})$. The states of $\mathcal{T}(\mathcal{M})$ are the configurations of $\mathcal{M}$, which consist of a state from $\mathcal{S}$, its tape contents, and the position of the read/write head. We denote by $\check{\mathcal{D}}_\Box = \{\check{d} \mid d \in \mathcal{D}_\Box\}$ the set of *marked* symbols; a *tape instance* is a sequence $\delta \in (\mathcal{D}_\Box \cup \check{\mathcal{D}}_\Box)^*$ such that δ contains exactly one element of the set of marked symbols $\check{\mathcal{D}}_\Box$, indicating the position of the read/write head. We adopt a convention to concisely denote an update of the placement of the tape head marker. Let δ be an element of $\mathcal{D}_\Box^*$. Then by $\delta^<$ we denote the element of $(\mathcal{D}_\Box \cup \check{\mathcal{D}}_\Box)^*$ obtained by placing the tape head marker on the right-most symbol of δ (if it exists), and $\check{\Box}$ otherwise. Similarly ${}^>\delta$ is obtained by placing the tape head marker on the left-most symbol of δ (if it exists), and $\check{\Box}$ otherwise.

Definition 10. *Let $\mathcal{M} = (\mathcal{S}, \longrightarrow, \uparrow, \downarrow)$ be an RTM. The* transition system $\mathcal{T}(\mathcal{M})$ associated with $\mathcal{M}$ *is defined as follows:*

1. *its set of states is the set $C_\mathcal{M} = \{(s,\delta) \mid s \in \mathcal{S},\ \delta$ a tape instance$\}$ of all configurations of $\mathcal{M}$;*
2. *its transition relation $\longrightarrow \subseteq C_\mathcal{M} \times \mathcal{A}_\tau \times C_\mathcal{M}$ is a relation satisfying, for all $a \in \mathcal{A}_\tau$, $d, e \in \mathcal{D}_\Box$ and $\delta_L, \delta_R \in \mathcal{D}_\Box^*$:*
   - $(s, \delta_L \check{d} \delta_R) \xrightarrow{a} (t, \delta_L{}^< e \delta_R)$ *iff* $s \xrightarrow{a[d/e]L} t$,
   - $(s, \delta_L \check{d} \delta_R) \xrightarrow{a} (t, \delta_L e^> \delta_R)$ *iff* $s \xrightarrow{a[d/e]R} t$;
3. *its initial state is the configuration $(\uparrow, \check{\Box})$; and*
4. *its set of final states is the set $\{(s,\delta) \mid \delta$ a tape instance, $s\downarrow\}$.*

Turing introduced his machines to define the notion of *effectively computable function* in [15]. By analogy, the notion of RTM can be used to define a notion of *effectively executable behaviour*.

Definition 11 (Executability)**.** *A transition system is* executable *if it is the transition system associated with some RTM.*

In the theory of executability, we use the notion of executable transition systems to evaluate the absolute expressiveness of process calculi in two aspects. On the one hand, if very transition system associated with a process expression specified in a process calculus is executable modulo some behavioural equivalence, then we say that the process calculus is *executable* modulo that behavioural equivalence. One the other hand, if every executable transition system is behavioural equivalent to some transition system associated with a process expression specified in a process calculus modulo some behavioural equivalence, then we say that the process calculus is *reactively Turing powerful* modulo that behavioural equivalence.

We only brief explain that both $\mathrm{TCP}^;$ and $\mathrm{TCP}^\sharp$ are executable. We observe that their transition systems are effective and does not contain any unbounded branching behaviour. Thus we can apply the result from [4] and draw a conclusion that they are executable modulo $\underline{\leftrightarrow}_{\mathrm{b}}^{\Delta}$.

Now we emphasis on showing that $\mathrm{TCP}^\sharp$ is a reactively Turing powerful process calculus modulo $\underline{\leftrightarrow}_{\mathrm{b}}^{\Delta}$.

We first introduce the notion of bisimulation up to $\underline{\leftrightarrow}_{\mathrm{b}}$, which is a useful tool to establish the proofs in this section. Note that we adopt a non-symmetric bisimulation up to relation.

Definition 12. *Let $T = (\mathcal{S}, \longrightarrow, \uparrow, \downarrow)$ a transition system. A relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a bisimulation up to $\underline{\leftrightarrow}_b$ if, whenever $s_1 \mathcal{R} s_2$, then for all $a \in \mathcal{A}_\tau$:*

1. *if $s_1 \longrightarrow^* s_1'' \xrightarrow{a} s_1'$, with $s_1 \underline{\leftrightarrow}_b s_1''$ and $a \neq \tau \vee s_1'' \not\underline{\leftrightarrow}_b s_1'$, then there exists s_2' such that $s_2 \xrightarrow{a} s_2'$, $s_1'' \underline{\leftrightarrow}_b \circ \mathcal{R}\ s_2$ and $s_1' \underline{\leftrightarrow}_b \circ \mathcal{R}\ s_2'$;*

2. *if* $s_2 \xrightarrow{a} s_2'$, *then there exist* s_1', s_1'' *such that* $s_1 \longrightarrow^* s_1'' \xrightarrow{a} s_1'$, $s_1'' \leftrightarrows_b s_1$ *and* $s_1' \leftrightarrows_b \circ \mathcal{R}\, s_2'$;
3. *if* $s_1 \downarrow$, *then there exists* s_2' *such that* $s_2 \longrightarrow^* s_2'$ *and* $s_2' \downarrow$; *and*
4. *if* $s_2 \downarrow$, *then there exists* s_1' *such that* $s_1 \longrightarrow^* s_1'$ *and* $s_1' \downarrow$;

Lemma 2. *If* $\mathcal{R}$ *is a bisimulation up to* $\leftrightarrows_b$, *then* $\mathcal{R} \subseteq \leftrightarrows_b$.

Proof. It is sufficient to prove that $\leftrightarrows_{\mathrm{b}} \circ \mathcal{R}$ is a branching bisimulation, for $\leftrightarrows_{\mathrm{b}}$ is an equivalence relation. Let $s_1, s_2, s_3 \in \mathcal{S}$ and $s_1 \leftrightarrows_{\mathrm{b}} s_2\, \mathcal{R}\, s_3$.

1. Suppose $s_1 \xrightarrow{a} s_1'$. We distinguish two cases:
   (a) If $a = \tau$ and $s_1 \leftrightarrows_{\mathrm{b}} s_1'$, then $s_1' \leftrightarrows_{\mathrm{b}} s_1 \leftrightarrows_{\mathrm{b}} s_2$, so $s_1' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R}\, s_3$. It satisfies Condition 1 of the definition of branching bisimulation.
   (b) Otherwise, we have $a \neq \tau \vee s_1 \not\leftrightarrows_{\mathrm{b}} s_1'$. Then, since $s_1 \leftrightarrows_{\mathrm{b}} s_2$, according to Definition 3, there exist s_2'' and s_2' such that $s_2 \longrightarrow^* s_2'' \xrightarrow{a} s_2'$, $s_1 \leftrightarrows_{\mathrm{b}} s_2''$ and $s_1' \leftrightarrows_{\mathrm{b}} s_2'$. Note that $s_2 \leftrightarrows_{\mathrm{b}} s_1 \leftrightarrows_{\mathrm{b}} s_2''$, and it is needed to apply Condition 1 of Definition 12. Then we have there exist s_4'', s_4' and s_3' such that $s_3 \xrightarrow{a} s_3'$ and $s_2'' \leftrightarrows_{\mathrm{b}} s_4''\, \mathcal{R}\, s_3$ and $s_2' \leftrightarrows_{\mathrm{b}} s_4'\, \mathcal{R}\, s_3'$. Since $s_1' \leftrightarrows_{\mathrm{b}} s_2' \leftrightarrows_{\mathrm{b}} s_4'$ and $s_4'\, \mathcal{R}\, s_3'$, it follows that $s_1' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R}\, s_3'$. It satisfies Condition 1 of the definition of branching bisimulation.
2. If $s_3 \xrightarrow{a} s_3'$, then according to Definition 12, there exist s_2'' and s_2' such that $s_2 \longrightarrow^* s_2'' \xrightarrow{a} s_2'$, $s_2'' \leftrightarrows_{\mathrm{b}} s_2$ and $s_2' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R}\, s_3'$, since $s_1 \leftrightarrows_{\mathrm{b}} s_2 \leftrightarrows_{\mathrm{b}} s_2''$ and $s_2'' \xrightarrow{a} s_2'$, by Definition 3, there exist s_1'' and s_1' such that $s_1 \longrightarrow^* s_1'' \xrightarrow{(a)} s_1'$ with $s_1'' \leftrightarrows_{\mathrm{b}} s_2''$ and $s_1' \leftrightarrows_{\mathrm{b}} s_2'$. Since $s_2'' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R}\, s_3$ and $s_2' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R}\, s_3'$, it follows that $s_1'' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R}\, s_3$ and $s_1' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R}\, s_3'$. It satisfies the symmetry of Condition 1 of the definition of branching bisimulation.

The termination condition is also satisfied from Definition 12.

Therefore, a branching bisimulation up to $\leftrightarrows_{\mathrm{b}}$ is included in $\leftrightarrows_{\mathrm{b}}$. □

Next we show that TCP$^\sharp$ is a reactively Turing powerful by writing a specification of the reactive Turing machine with TCP$^\sharp$modulo $\leftrightarrows_{\mathrm{b}}^{\Delta}$. The proof consists of five steps.

1. We first write the specification of a terminating half counter;
2. then we show that every regular process can be specified in TCP$^\sharp$;
3. next we use two half counters and a regular process to encode a terminating stack;
4. with two stacks and a regular process we can specify a tape; and
5. finally we use a tape and a regular control process to specify an RTM.

We first recall the following infinite specification in TSP$^;$ of a terminating half counter:

$$\begin{aligned} C_n &= a.C_{n+1} + b.B_n + \mathbf{1}\,(n \in \mathbb{N}) \\ B_n &= a.B_{n-1} + \mathbf{1}\,(n \geq 1) \\ B_0 &= c.C_0 + \mathbf{1} \end{aligned}$$

We provide a specification of a counter in TCP$^\sharp$ as follows:

$$HC = ((a + \mathbf{1})^\sharp (b + \mathbf{1}); (c + \mathbf{1}))^*$$

We have the following lemma:

Lemma 3. $C_0 \leftrightarrows_b^{\Delta} HC$

Proof. We verify that $HC \leftrightarrows_{\mathrm{b}}^{\Delta} C_0$. Consider the following relation:

$$\mathcal{R}_1 = \{(C_0, HC)\} \cup \{(C_n, (a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1})^n; (c+\mathbf{1}); HC) \mid n \geq 1\} \cup \{(B_n, (a+\mathbf{1})^n; (c+\mathbf{1}); HC) \mid n \in \mathbb{N}\} \ .$$

We let $\mathcal{R}_2$ be the symmetrical relation of $\mathcal{R}_1$. We show that $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ is a divergence-preserving branching bisimulation as follows:

Note that $\mathcal{R}$ satisfies the divergence-preserving condition since there is no infinite sequence of τ transitions. In this prove, we only illustrate the pairs in $\mathcal{R}_1$, since we can use the symmetrical argument for the pairs in $\mathcal{R}_2$. We first consider the pair (C_0, HC). Note that C_0 has the following transitions:

$$C_0 \xrightarrow{a} C_1, \text{ and}$$
$$C_0 \xrightarrow{b} B_0 \ ,$$

which are simulated by:

$$HC \xrightarrow{a} (a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1}); (c+\mathbf{1}); HC, \text{ and}$$
$$HC \xrightarrow{b} (c+\mathbf{1}); HC \ ,$$

with $(C_1, (a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1}); (c+\mathbf{1}); HC) \in \mathcal{R}$ and $(B_0, (c+\mathbf{1}); HC) \in \mathcal{R}$. Moreover, we have $C_0 \downarrow$ and $HC \downarrow$.

Now we consider the pair $(C_n, (a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1})^n; (c+\mathbf{1}); HC)$, with $n \geq 1$. Note that C_n has the following transitions:

$$C_n \xrightarrow{a} C_{n+1}, \text{ and}$$
$$C_n \xrightarrow{b} B_n \ ,$$

which are simulated by:

$$(a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1})^n; (c+\mathbf{1}); HC \xrightarrow{a} (a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1})^{n+1}; (c+\mathbf{1}); HC, \text{ and}$$
$$(a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1})^n; (c+\mathbf{1}); HC \xrightarrow{b} (a+\mathbf{1})^n; (c+\mathbf{1}); HC \ ,$$

with $(C_{n+1}, (a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1})^{n+1}; (c+\mathbf{1}); HC) \in \mathcal{R}$ and $(B_n, (a+\mathbf{1})^n; (c+\mathbf{1}); HC) \in \mathcal{R}$. Moreover, we have $C_n \downarrow$ and $(a+\mathbf{1})^{\sharp}(b+\mathbf{1}); (a+\mathbf{1})^n; (c+\mathbf{1}); HC \downarrow$.

Now we proceed to consider the pair $(B_0, (c+\mathbf{1}); HC)$. Note that B_0 has the following transition:

$$B_0 \xrightarrow{c} C_0 \ ,$$

which is simulated by:

$$(c+\mathbf{1}); HC \xrightarrow{c} HC \ ,$$

with $(C_0, HC) \in \mathcal{R}$. Moreover, we have $B_0 \downarrow$ and $(c+\mathbf{1}); HC \downarrow$.

Next we consider the pair $(B_n, (a+\mathbf{1})^n; (c+\mathbf{1}); HC)$, with $n \geq 1$. Note that B_n has the following transition:

$$B_n \xrightarrow{a} B_{n-1} \ ,$$

which is simulated by:

$$(a+\mathbf{1})^n;(c+\mathbf{1});HC \xrightarrow{a} (a+\mathbf{1})^{n-1};(c+\mathbf{1});HC \ ,$$

with $(B_{n-1},(a+\mathbf{1})^{n-1};(c+\mathbf{1});HC) \in \mathcal{R}$. Moreover, we have $B_n\downarrow$ and $(a+\mathbf{1})^n;(c+\mathbf{1});HC\downarrow$.

Hence, we have $C_0 \leftrightarrows^{\Delta}_{\mathrm{b}} HC$. □

Next we show that every regular process can be specified in TCP$^\sharp$ modulo $\leftrightarrows^{\Delta}_{\mathrm{b}}$. A regular process with at finite set of action labels $\mathcal{A}_\tau$ is given by $P_i = \sum_{j=1}^n \alpha_{ij};P_j+\beta_i\,(i=1,\ldots,n)$ where α_{ij} and β_i are finite sums of actions from $\mathcal{A}_\tau$. We show the following lemma.

Lemma 4. *Every regular process can be specified in TCP$^\sharp$ modulo $\leftrightarrows^{\Delta}_{b}$.*

Proof. We consider a regular process with at finite set of action labels $\mathcal{A}_\tau$ which is given by $P_i = \sum_{j=1}^n \alpha_{ij};P_j+\beta_i\,(i=1,\ldots,n)$ where α_{ij} and β_i are finite sums of actions from $\mathcal{A}_\tau$. We let $c!0,c!1,\ldots,c!(n+1),c?0,c?1,\ldots,c?(n+1)$ be labels that are not in $\mathcal{A}_\tau$.

Consider the following process:

$$\begin{aligned}
G_i &= \sum_{j=1}^{n}\alpha_{ij};(c!j+\mathbf{1})+\beta_i;(c!0+\mathbf{1})\\
M &= \left(\sum_{j=1}^{n}(c?j+\mathbf{1});G_j+(c!(n+1)+\mathbf{1});(c?(n+1)+\mathbf{1})\right)^{\sharp}(c?0+\mathbf{1})\\
N &= \left(\sum_{j=1}^{n+1}(c?j+\mathbf{1});(c!j+\mathbf{1})\right)^{\sharp}((c?0+\mathbf{1});(c!0+\mathbf{1}))
\end{aligned}$$

Note that ; is associative and we suppose that ; binds stronger than +. We verify that $P_i \leftrightarrows^{\Delta}_{\mathrm{b}} [G_i;M \parallel N]_{\{c\}}$. We let $Q = \left(\sum_{j=1}^{n}(c?j+\mathbf{1});G_j+(c!(n+1)+\mathbf{1});(c?(n+1)+\mathbf{1})\right)$ and $O = \left(\sum_{j=1}^{n+1}(c?j+\mathbf{1});(c!j+\mathbf{1})\right)$. We let

$$\begin{aligned}
\mathcal{R}_1 &= \{(P_i,[G_i;M;Q^k \parallel N;O^k]_{\{c\}}) \mid k\in\mathbb{N}, i=1,\ldots,n\}\\
&\cup \{(P_i,[(c!i+\mathbf{1});M;Q^k \parallel N;O^k]_{\{c\}}) \mid k\in\mathbb{N}, i=1,\ldots,n\}\\
&\cup \{(P_i,[M;Q^k \parallel (c!i+\mathbf{1});N;O^{k+1}]_{\{c\}}) \mid k\in\mathbb{N}, i=1,\ldots,n\}\\
&\cup \{(\mathbf{1},[(c!0+\mathbf{1});M;Q^k \parallel N;O^k]_{\{c\}}) \mid k\in\mathbb{N}\}\\
&\cup \{(\mathbf{1},[M;Q^k \parallel (c!0+\mathbf{1});O^k]_{\{c\}}) \mid k\in\mathbb{N}\}\\
&\cup \{(\mathbf{1},[Q^k \parallel O^k]_{\{c\}}) \mid k\in\mathbb{N}\}\\
&\cup \{(\mathbf{1},[(c?(n+1)+\mathbf{1});Q^k \parallel (c!(n+1)+\mathbf{1});O^k]_{\{c\}}) \mid k\in\mathbb{N}\}\ ;
\end{aligned}$$

and we let $\mathcal{R}_2$ be the symmetrical relation of $\mathcal{R}_1$. We show that $\mathcal{R}=\mathcal{R}_1\cup\mathcal{R}_2$ is a divergence-preserving branching bisimulation. We shall only verify the pairs in $\mathcal{R}_1$ in this proof since $\mathcal{R}$ is symmetrical.

For the set of pairs $\{(P_i,[G_i;M;Q^k \parallel N;O^k]_{\{c\}}) \mid k\in\mathbb{N}, i=1,\ldots,n\}$, note that P_i has the following transitions: $P_i \xrightarrow{a} P_j$ if a is a summand of α_{ij}, or $P_i \xrightarrow{a} \mathbf{1}$ if a is a summand of β_j.

The first transition is simulated by the following transitions:

$$\begin{aligned}
&[G_i;M;Q^k \parallel N;O^k]_{\{c\}} \xrightarrow{a} [(c!j+\mathbf{1});M;Q^k \parallel N;O^k]_{\{c\}}\\
&\xrightarrow{\tau} [M;Q^k \parallel (c!j+\mathbf{1});N;O^{k+1}]_{\{c\}}\\
&\xrightarrow{\tau} [G_j;M;Q^{k+1} \parallel N;O^{k+1}]_{\{c\}}\ .
\end{aligned}$$

If $k \geq 1$, then the second transition is simulated by the following transitions:

$$\begin{array}{l}[G_i; M; Q^k \parallel N; O^k]_{\{c\}} \xrightarrow{a} [(c!0 + \mathbf{1}); M; Q^k \parallel N; O^k]_{\{c\}} \\ \xrightarrow{\tau} [M; Q^k \parallel (c!0 + \mathbf{1}); O^k]_{\{c\}} \xrightarrow{\tau} [Q^k \parallel O^k]_{\{c\}} \\ \xrightarrow{\tau} [(c?(n+1) + \mathbf{1}); Q^{k-1} \parallel (c!(n+1) + \mathbf{1}); O^{k-1}]_{\{c\}} \xrightarrow{\tau} [Q^{k-1} \parallel O^{k-1}]_{\{c\}} \\ \longrightarrow^* \mathbf{1} \ ;\end{array}$$

otherwise, if $k = 0$, then the second transition are simulated by:

$$\begin{array}{l}[G_i; M \parallel N]_{\{c\}} \xrightarrow{a} [(c!0 + \mathbf{1}); M \parallel N]_{\{c\}} \\ \xrightarrow{\tau} [M \parallel (c!0 + \mathbf{1})]_{\{c\}} \xrightarrow{\tau} \mathbf{1} \ .\end{array}$$

We have that that $(P_j, [(c!j+\mathbf{1}); M; Q^k \parallel N; O^k]_{\{c\}}) \in \mathcal{R}$, $(P_j, [M; Q^k \parallel (c!j+\mathbf{1}); N; O^{k+1}]_{\{c\}}) \in \mathcal{R}$, $(P_j, [G_j; M; Q^{k+1} \parallel N; O^{k+1}]_{\{c\}}) \in \mathcal{R}$, $(\mathbf{1}, [(c!0 + \mathbf{1}); M; Q^k \parallel N; O^k]_{\{c\}}) \in \mathcal{R}$, $(\mathbf{1}, [M; Q^k \parallel (c!0 + \mathbf{1}); O^k]_{\{c\}}) \in \mathcal{R}$, $(\mathbf{1}, [Q^k \parallel O^k]_{\{c\}})$, $(\mathbf{1}, [(c?(n+1) + \mathbf{1}); Q^k \parallel (c!(n+1) + \mathbf{1}); O^k]_{\{c\}}) \in \mathcal{R}$ and $(\mathbf{1}, \mathbf{1}) \in \mathcal{R}$ for all $k \in \mathbb{N}$ and $i, j = 1, \ldots, n$.

One can easily verify that all the other pairs satisfy the condition of branching bisimulation. The relation $\mathcal{R}$ also satisfies the divergence-preserving condition since no infinite τ-transition sequence is allowed from any process defined in $\mathcal{R}$.

Therefore, we get a finite specification of every regular process in $\mathrm{TCP}^\sharp$ modulo $\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}$. □

Now we show that a stack can be specified by a regular process and two half counters. We first give an infinite specification in $\mathrm{TSP}^;$ of a stack as follows:

$$\begin{array}{rcl} S_\epsilon & = & \Sigma_{d \in \mathcal{D}_\square} push?d.S_d + pop!\square.S_\epsilon + \mathbf{1} \\ S_{d\delta} & = & pop!d.S_\delta + \Sigma_{e \in \mathcal{D}_\square} push?e.S_{ed\delta} + \mathbf{1} \ . \end{array}$$

Note that $\mathcal{D}_\square$ is a finite set of symbols. We suppose that $\mathcal{D}_\square$ contains N symbols (including $\square$). We use ϵ to denote empty sequence. We first define an encoding from sequence of symbols to natural numbers $\lceil \_ \rceil : {\mathcal{D}_\square}^* \Rightarrow \mathbb{N}$ inductively defined as follows:

$$\begin{array}{rcl} \lceil \epsilon \rceil & = & 0 \\ \lceil d_k \rceil & = & k (k = 1, 2, \ldots, N) \\ \lceil d_k \sigma \rceil & = & k + N \times \lceil \sigma \rceil \ . \end{array}$$

We define a stack in TCP$^\sharp$ as follows:

$$\begin{aligned}
S &= [X_\emptyset \parallel P_1 \parallel P_2]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}}\\
P_j &= ((a_j!a+\mathbf{1})^\sharp(b_j!b+\mathbf{1});(c_j!c+\mathbf{1}))^*\,(j=1,2)\\
X_\emptyset &= (\Sigma_{j=1}^N((push?d_j+\mathbf{1});(a_1?a+\mathbf{1})^j;(b_1+\mathbf{1});X_j)+pop!\Box)^*\\
X_k &= \Sigma_{j=1}^N((push?d_j+\mathbf{1});Push_j)+(pop!d_k+\mathbf{1});Pop_k\,(k=1,2,\ldots,N)\\
Push_k &= Shift1to2;(a_1?a+\mathbf{1})^k;NShift2to1;X_k\,(k=1,2,\ldots,N)\\
Pop_k &= (a_1?a+\mathbf{1})^k;1/NShift1to2;Test_\emptyset\\
Shift1to2 &= ((a_1?a+\mathbf{1});(a_2?a+\mathbf{1}))^*;(c_1?c+\mathbf{1});(b_2?b+\mathbf{1})\\
NShift2to1 &= ((a_2?a+\mathbf{1});(a_1?a+\mathbf{1})^N)^*;(c_2?c+\mathbf{1});(b_1?b+\mathbf{1})\\
1/NShift1to2 &= ((a_1?a+\mathbf{1})^N;(a_2?a+\mathbf{1}))^*;(c_1?c+\mathbf{1});(b_2?b+\mathbf{1})\\
Test_\emptyset &= (a_2?a+\mathbf{1});(a_1?a+\mathbf{1});Test_1+(c_2?c+\mathbf{1});X_\emptyset\\
Test_1 &= (a_2?a+\mathbf{1});(a_1?a+\mathbf{1});Test_2+(c_2?c+\mathbf{1});X_1\\
Test_2 &= (a_2?a+\mathbf{1});(a_1?a+\mathbf{1});Test_3+(c_2?c+\mathbf{1});X_2\\
&\ldots\\
Test_N &= (a_2?a+\mathbf{1});(a_1?a+\mathbf{1});Test_1+(c_2?c+\mathbf{1});X_N\ .
\end{aligned}$$

We have the following result.

Lemma 5. $S_\epsilon \leftrightarrows_b^\Delta S$.

Proof. We define some auxiliary process:

$$\begin{aligned}
P_j(0) &= ((a_j!a+\mathbf{1})^\sharp(b_j!b+\mathbf{1});(c_j!c+\mathbf{1}))^*\,(j=1,2)\\
P_j(n) &= (a_j!a+\mathbf{1})^\sharp(b_j!b+\mathbf{1});(a_j!a+\mathbf{1})^n;(c_j!c+\mathbf{1}));P_j,\,(j=1,2;\,n=1,2,\ldots)\\
Q_j(n) &= (a_j!a+\mathbf{1})^n;(c_j!c+\mathbf{1}));P_j,\,(j=1,2;\,n\in\mathbb{N})\ .
\end{aligned}$$

P_0 and P_1 behave as two half counters.

We let $\mathcal{R}_1=\{(S_\epsilon,S)\}\cup\{(S_{d_j\delta},[X_j;X_\epsilon\parallel Q_1(m)\parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}})\mid j=\lceil d_j\rceil,m=\lceil d_j\delta\rceil,d\in\mathcal{D}_\Box,\delta\in\mathcal{D}_\Box^*\}$. We let $\mathcal{R}_2$ be the symmetrical relation of $\mathcal{R}_1$. We verify that $\mathcal{R}=\mathcal{R}_1\cup\mathcal{R}_2\cup\leftrightarrows_{\rm b}^\Delta$ is a divergence-preserving branching bisimulation relation.

Note that S_ϵ has the following transitions:

$$\begin{aligned}
&S_\epsilon \xrightarrow{push?d_j} S_{d_j} \text{ for all } j=1,2,\ldots,N, \text{ and}\\
&S_\epsilon \xrightarrow{pop!\Box} S_\epsilon\ .
\end{aligned}$$

They are simulated by the following transitions:

$$\begin{aligned}
&S \xrightarrow{push?d_j} [(a_1?a+\mathbf{1})^j;(b_1+\mathbf{1});X_j;X_\epsilon\parallel P_1(0)\parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}}\\
&\longrightarrow^* [(b_1+\mathbf{1});X_j;X_\epsilon\parallel P_1(j)\parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}}\\
&\longrightarrow^* [X_j;X_\epsilon\parallel Q_1(j)\parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \text{ for all } j=1,2,\ldots,N, \text{ and}\\
&S \xrightarrow{pop!\Box} S\ .
\end{aligned}$$

We only consider the first case, since the second transition is trivial. We have $(S_{d_j}, [X_j; X_\epsilon \parallel Q_1(j) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}}) \in \mathcal{R}$. We denote the sequence of transitions $[(a_1?a+\mathbf{1})^j; (b_1+\mathbf{1}); X_j; X_\epsilon \parallel P_1(0) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \longrightarrow^* [X_j; X_\epsilon \parallel Q_1(j) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}}) \in \mathcal{R}$ by $s_0 \longrightarrow^* s_m$. It is obvious that $s_0 \leftrightarrows_{\mathrm{b}}^{\Delta} \ldots s_m$. Therefore, $S \overset{push?d_j}{\longrightarrow} s_0$, and $s_0 \leftrightarrows_{\mathrm{b}}^{\Delta} s_m$ with $(S_{d_j}, s_m) \in \mathcal{R}$.

Note that $S_{d_j\delta}$ has the following transitions:

$$S_{d_j\delta} \overset{push?d_k}{\longrightarrow} S_{d_kd_j\delta} \text{ for all } k = 1,2,\ldots,N, \text{ and}$$
$$S_{d_j\delta} \overset{pop!d_j}{\longrightarrow} S_{d_k\delta'}, \text{ where } d_k\delta' = \delta \ .$$

They are simulated by the following transitions:

$$\begin{aligned}
&[X_j; X_\epsilon \parallel Q_1(\lceil d_j\delta \rceil) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \overset{push?d_k}{\longrightarrow} [Push_k; X_\epsilon \parallel Q_1(\lceil d_j\delta \rceil) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \\
&\longrightarrow^* [(a_1?a+\mathbf{1})^k; NShift2to1; X_k; X_\epsilon \parallel P_1(0) \parallel Q_2(\lceil d_j\delta \rceil)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \\
&\longrightarrow^* [NShift2to1; X_k; X_\epsilon \parallel P_1(\lceil d_k \rceil) \parallel Q_2(\lceil d_j\delta \rceil)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \\
&\longrightarrow^* [X_k; X_\epsilon \parallel Q_1(\lceil d_kd_j\delta \rceil) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \text{ for all } d_j, d_k \in \mathcal{D}_\Box, \delta \in \mathcal{D}_\Box^* \text{ and} \\
&[X_j; X_\epsilon \parallel Q_1(\lceil d_j\delta \rceil) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \overset{pop!d_j}{\longrightarrow} [Pop_j; X_\epsilon \parallel Q_1(\lceil d_j\delta \rceil) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \\
&\longrightarrow^* [1/NShift1to2; Test_\emptyset; X_\epsilon \parallel Q_1(\lceil d_j\delta \rceil - k) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \\
&\longrightarrow^* [Test_\emptyset; X_\epsilon \parallel P_1(0) \parallel Q_2(\lceil \delta \rceil)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \\
&\longrightarrow^* [X_k; X_\epsilon \parallel Q_1(\lceil d_k\delta' \rceil \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \text{ for all } d_j \in \mathcal{D}_\Box, \delta \in \mathcal{D}_\Box^* \text{ and } \delta = d_k\delta' \ .
\end{aligned}$$

We have $(S_{d_kd_j\delta}, [X_k; X_\epsilon \parallel Q_1(\lceil d_kd_j\delta \rceil) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}}) \in \mathcal{R}$ and $(S_{d_k\delta'}, [X_k; X_\epsilon \parallel Q_1(\lceil d_k\delta' \rceil \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}}) \in \mathcal{R}$. By using a similar analysis with the previous case, we conclude that $\mathcal{R}$ is a bisimulation up to $\leftrightarrows_{\mathrm{b}}$. By Lemma 2, we have $\mathcal{R} \subseteq \leftrightarrows_{\mathrm{b}}$. Moreover, there is no infinite τ-transition sequence from any process defined above. Therefore, $\mathcal{R} \subseteq \leftrightarrows_{\mathrm{b}}^{\Delta}$.

Hence, we have $S_\epsilon \leftrightarrows_{\mathrm{b}}^{\Delta} S$. □

Next we proceed to define the tape by means of two stacks. We consider the following infinite specification in TSP$^;$ of a tape:

$$T_{\delta_L\check{d}\delta_R} = r!d.T_{\delta_L\check{d}\delta_R} + \Sigma_{e\in\mathcal{D}_\Box} w?e.T_{\delta_L\check{e}\delta_R} + L?m.T_{\delta_L{}^{<}d\delta_R} + R?m.T_{\delta_Ld^{>}\delta_R} + \mathbf{1} \ .$$

We define the tape process in TCP$^\sharp$ as follows:

$$\begin{aligned}
T &= [T_\Box \parallel S_1 \parallel S_2]_{\{push_1,pop_1,push_2,pop_2\}} \\
T_d &= r!d.T_d + \Sigma_{e\in\mathcal{D}_\Box} w?e.T_e + L?m.Left_d + R?m.Right_d + \mathbf{1}(d \in \mathcal{D}_\Box) \\
Left_d &= \Sigma_{e\in\mathcal{D}_\Box}((pop_1?e+\mathbf{1}); (push_2!d+\mathbf{1}); T_e) \\
Right_d &= \Sigma_{e\in\mathcal{D}_\Box}((pop_2?e+\mathbf{1}); (push_1!d+\mathbf{1}); T_e) \ ,
\end{aligned}$$

where S_1 and S_2 are two stacks with $push_1, pop_1, push_2$ and pop_2 as their interfaces.

We establish the following result.

Lemma 6. $T_{\check{\Box}} \leftrightarrows_b^{\Delta} T$.

Proof. We define the following auxiliary processes:

$$\begin{aligned} S_1(\delta) &= [X_{1,k} \parallel Q_1(\lceil\delta\rceil) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}} \\ S_2(\delta) &= [X_{2,k} \parallel Q_1(\lceil\delta\rceil) \parallel P_2(0)]_{\{a_1,a_2,b_1,b_2,c_1,c_2\}}, \text{where}\, \delta = d_k\delta' \ . \end{aligned}$$

$X_{1,k}$ and $X_{2,k}$ is obtained by renaming *push* and *pop* in X_k to $push_1, pop_1, push_2$ and pop_2 respectively. We use $\overline{\delta}$ to denote the reverse sequence of δ.

We verify that

$$\mathcal{R} = \{(T_{\delta_L \check{d} \delta_R}, [T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}}) \mid d \in \mathcal{D}_\Box, \delta_L, \delta_R \in \mathcal{D}_\Box^*\} \subseteq \underline{\leftrightarrow}_{\mathrm{b}}^{\Delta} \ .$$

$T_{\delta_L \check{d} \delta_R}$ has the following transitions:

$$\begin{aligned} T_{\delta_L \check{d} \delta_R} &\xrightarrow{r!d} T_{\delta_L \check{d} \delta_R} \\ T_{\delta_L \check{d} \delta_R} &\xrightarrow{w?e} T_{\delta_L \check{e} \delta_R} \text{ for all } e \in \mathcal{D}_\Box \\ T_{\delta_L \check{d} \delta_R} &\xrightarrow{L?m} T_{\delta_L{}^{<} d \delta_R} \text{ if } \delta_L \neq \epsilon \\ T_{\delta_L \check{d} \delta_R} &\xrightarrow{R?m} T_{\delta_L d^{>} \delta_R} \text{ if } \delta_R \neq \epsilon \\ T_{\delta_L \check{d} \delta_R} &\xrightarrow{L?m} T_{\epsilon \check{\Box} d \delta_R} \text{ if } \delta_L = \epsilon \text{ and} \\ T_{\delta_L \check{d} \delta_R} &\xrightarrow{R?m} T_{\delta_L d \check{\Box} \epsilon} \text{ if } \delta_R = \epsilon \ . \end{aligned}$$

They are simulated by the following transitions:

$$\begin{aligned} &[T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \xrightarrow{r!d} [T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \\ &[T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \xrightarrow{e?d} [T_e \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \text{ for all } e \in \mathcal{D}_\Box \\ &[T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \xrightarrow{L?m} [Left_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \\ &\longrightarrow^* [T_e \parallel S_1(\overline{\delta'_L}) \parallel S_2(d\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}}, \delta_L = \delta'_L e, \text{ if } \delta_L \neq \epsilon \\ &[T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \xrightarrow{R?m} [Right_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \\ &\longrightarrow^* [T_e \parallel S_1(\overline{\delta_L d}) \parallel S_2(\delta'_R)]_{\{push_1,pop_1,push_2,pop_2\}}, \delta_R = e\delta_R, \text{ if } \delta_R \neq \epsilon \\ &[T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \xrightarrow{L?m} [Left_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \\ &\longrightarrow^* [T_\Box \parallel S_1(\epsilon) \parallel S_2(d\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}}, \text{ if } \delta_L = \epsilon \\ &[T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \xrightarrow{R?m} [Right_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}} \\ &\longrightarrow^* [T_\Box \parallel S_1(\overline{\delta_L d}) \parallel S_2(\epsilon)]_{\{push_1,pop_1,push_2,pop_2\}}, \text{ if } \delta_R = \epsilon \ . \end{aligned}$$

We have

$$\begin{aligned} &(T_{\delta_L \check{d} \delta_R}, [T_d \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}}) \in \mathcal{R}, \\ &(T_{\delta_L \check{e} \delta_R}, [T_e \parallel S_1(\overline{\delta_L}) \parallel S_2(\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}}) \in \mathcal{R}, \\ &(T_{\delta_L{}^{<} d \delta_R}, [T_e \parallel S_1(\overline{\delta'_L}) \parallel S_2(d\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}}) \in \mathcal{R}, \\ &(T_{\delta_L d^{>} \delta_R}, [T_e \parallel S_1(\overline{\delta_L d}) \parallel S_2(\delta'_R)]_{\{push_1,pop_1,push_2,pop_2\}}) \in \mathcal{R}, \\ &(T_{\epsilon \check{\Box} d \delta_R}, [T_\Box \parallel S_1(\epsilon) \parallel S_2(d\delta_R)]_{\{push_1,pop_1,push_2,pop_2\}}) \in \mathcal{R}, \text{ and} \\ &(T_{\delta_L d \check{\Box} \epsilon}, [T_\Box \parallel S_1(\overline{\delta_L d}) \parallel S_2(\epsilon)]_{\{push_1,pop_1,push_2,pop_2\}}) \in \mathcal{R} \ . \end{aligned}$$

By an analysis similar from Lemma 5, we have $\mathcal{R}$ is a bisimulation up to $\leftrightarrows_{\mathrm{b}}$. Therefore, $\mathcal{R} \subset \leftrightarrows_{\mathrm{b}}$. Moreover, there is no infinite τ-transition sequence from the processes defined above. Therefore, $\mathcal{R} \subseteq \leftrightarrows_{\mathrm{b}}^{\Delta}$.

Hence, we have $T_{\check{\square}} \leftrightarrows_{\mathrm{b}}^{\Delta} T$. □

Finally, we construct a finite control process for an RTM $\mathcal{M} = (\mathcal{S}_{\mathcal{M}}, \longrightarrow_{\mathcal{M}}, \uparrow_{\mathcal{M}}, \downarrow_{\mathcal{M}})$ as follows:

$$C_{s,d} = \Sigma_{(s,d,a,e,M,t)\in\longrightarrow_{\mathcal{M}}}(a.w!e.M!m.\Sigma_{f\in\mathcal{D}_{\square}} r?f.C_{t,f})[+\mathbf{1}]_{s\downarrow_{\mathcal{M}}} (s \in \mathcal{S}_{\mathcal{M}}, d \in \mathcal{D}_{\square}) \ .$$

We prove the following lemma.

Lemma 7. $\mathcal{T}(\mathcal{M}) \leftrightarrows_{b}^{\Delta} [C_{\uparrow_{\mathcal{M}},\square} \parallel T]_{\{r,w,L,R\}}$.

Proof. By the proof of Theorem 1, $\leftrightarrows_{\mathrm{b}}^{\Delta}$ is compatible with parallel composition. Therefore, it is enough to show that $\mathcal{T}(\mathcal{M}) \leftrightarrows_{\mathrm{b}}^{\Delta} [C_{\uparrow,\square} \parallel T_{\check{\square}}]_{\{r,w,L,R\}}$.

We define a binary relation $\mathcal{R}$ by:

$$\begin{aligned}
\mathcal{R} \;=\; & \{((s,\delta_L \check{d} \delta_R), [C_{s,d} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}}) \mid s \in \mathcal{S}_M, \delta_L, \delta_R \in \mathcal{D}_{\square}^*, d \in \mathcal{D}_{\square}\} \\
\cup\; & \{((s,\delta_L{}^{<} d \delta_R), [C_{s,f} \parallel T_{\delta_L{}^{<} d \delta_R}]_{\{r,w,L,R\}}) \mid s \in \mathcal{S}_M, \delta_L, \delta_R \in \mathcal{D}_{\square}^*, d \in \mathcal{D}_{\square}, \delta_L \neq \epsilon, \delta_L = \delta'_L f\} \\
\cup\; & \{((s,\delta_L d^{>} \delta_R), [C_{s,f} \parallel T_{\delta_L d^{>} \delta_R}]_{\{r,w,L,R\}}) \mid s \in \mathcal{S}_M, \delta_L, \delta_R \in \mathcal{D}_{\square}^*, d \in \mathcal{D}_{\square}, \delta_R \neq \epsilon, \delta_R = f \delta'_R\} \\
\cup\; & \{((s,\check{\square}\delta_R), [C_{s,\square} \parallel T_{\check{\square}\delta_R}]_{\{r,w,L,R\}}) \mid s \in \mathcal{S}_M, \delta_R \in \mathcal{D}_{\square}^*\} \\
\cup\; & \{((s,\delta_L \check{\square}), [C_{s,\square} \parallel T_{\delta_L \check{\square}}]_{\{r,w,L,R\}}) \mid s \in \mathcal{S}_M, \delta_L \in \mathcal{D}_{\square}^*\} \ .
\end{aligned}$$

We show that $\mathcal{R} \subseteq \leftrightarrows_{\mathrm{b}}^{\Delta}$.

$(s, \delta_L \check{d} \delta_R)$ has the following transitions:

$$\begin{aligned}
(s,\delta_L \check{d} \delta_R) &\xrightarrow{a} (t, \delta_L{}^{<} e \delta_R) \quad \text{if } (s,d,a,e,L,t) \in \longrightarrow_{\mathcal{M}}, \delta_L \neq \epsilon \\
(s,\delta_L \check{d} \delta_R) &\xrightarrow{a} (t, \delta_L e^{>} \delta_R) \quad \text{if } (s,d,a,e,R,t) \in \longrightarrow_{\mathcal{M}}, \delta_R \neq \epsilon \\
(s,\delta_L \check{d} \delta_R) &\xrightarrow{a} (t, \check{\square} e \delta_R) \quad \text{if } (s,d,a,e,L,t) \in \longrightarrow_{\mathcal{M}}, \delta_L = \epsilon \\
(s,\delta_L \check{d} \delta_R) &\xrightarrow{a} (t, \delta_L e \check{\square}) \quad \text{if } (s,d,a,e,R,t) \in \longrightarrow_{\mathcal{M}}, \delta_R = \epsilon \ .
\end{aligned}$$

They are simulated by:

$$\begin{aligned}
&[C_{s,d} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}} \xrightarrow{a} [w!e.L!m.\Sigma_{f\in\mathcal{D}_{\square}} r?f.C_{t,f} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}} \\
&\longrightarrow^* [C_{t,f} \parallel T_{\delta_L{}^{<} d \delta_R}]_{\{r,w,L,R\}}, \text{if } (s,d,a,e,L,t) \in \longrightarrow_{\mathcal{M}}, \delta_L \neq \epsilon, \delta_L = \delta'_L f \\
&[C_{s,d} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}} \xrightarrow{a} [w!e.R!m.\Sigma_{f\in\mathcal{D}_{\square}} r?f.C_{t,f} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}} \\
&\longrightarrow^* [C_{t,f} \parallel T_{\delta_L d^{>} \delta_R}]_{\{r,w,L,R\}}, \text{if } (s,d,a,e,R,t) \in \longrightarrow_{\mathcal{M}}, \delta_R \neq \epsilon, \delta_R = f \delta'_R \\
&[C_{s,d} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}} \xrightarrow{a} [w!e.L!m.\Sigma_{f\in\mathcal{D}_{\square}} r?f.C_{t,f} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}} \\
&\longrightarrow^* [C_{t,\square} \parallel T_{\check{\square} d \delta_R}]_{\{r,w,L,R\}}, \text{if } (s,d,a,e,L,t) \in \longrightarrow_{\mathcal{M}}, \delta_L = \epsilon \\
&[C_{s,d} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}} \xrightarrow{a} [w!e.R!m.\Sigma_{f\in\mathcal{D}_{\square}} r?f.C_{t,f} \parallel T_{\delta_L \check{d} \delta_R}]_{\{r,w,L,R\}} \\
&\longrightarrow^* [C_{t,\square} \parallel T_{\delta_L d \check{\square}}]_{\{r,w,L,R\}}, \text{if } (s,d,a,e,L,t) \in \longrightarrow_{\mathcal{M}}, \delta_R = \epsilon \ .
\end{aligned}$$

We apply similar analysis to other pairs in $\mathcal{R}$. Using the proof strategy similar to Lemma 5, it is straightforward show that $\mathcal{R}$ is a bisimulation up to $\leftrightarrows_{\mathrm{b}}$. Hence, we have $\mathcal{R} \subset \leftrightarrows_{\mathrm{b}}$. Moreover, using a similar strategy in the proof showing a π-calculus is reactively Turing powerful [**?**], we can show that $\mathcal{R}$

satisfies the divergence-preserving condition. For every infinite τ-transition sequence in $\mathcal{T}(\mathcal{M})$, we can find an infinite τ-transition sequence in the transition system induced from $[C_{\uparrow_{\mathcal{M}},\Box} \parallel T]_{\{r,w,L,R\}}$. Therefore, $\mathcal{R} \subset \leftrightarrows_{\mathrm{b}}^{\Delta}$.

Hence, we have $\mathcal{T}(\mathcal{M}) \leftrightarrows_{\mathrm{b}}^{\Delta} [C_{\uparrow_{\mathcal{M}},\Box} \parallel T]_{\{r,w,L,R\}}$. □

We have the following theorem.

Theorem 3. *TCP$^\sharp$ is reactively Turing powerful modulo $\leftrightarrows_b^{\Delta}$.*

7 Conclusion

In this paper we have proposed a revised operational semantics of the sequential composition operator in the presence of intermediate termination. We established two results which is still unsolved with the standard version of the sequential composition operator. We first proved that, with the revised semantics, every context-free process corresponds to a pushdown process modulo strong bisimilarity. We also proved that, TCP$^\sharp$ is a reactively Turing powerful process calculi modulo divergence-preserving branching bisimilarity.

There are still some negative premise in the revised sequential composition operator. For instance, unguarded recursion causes problems. Consider the following process:

$$P_1 = P_1; P_2 + \mathbf{1} \quad P_2 = a.\mathbf{1} \ .$$

According to our operational semantics, no transition is allowed from P_1. If we replace ; by $\cdot$, P_1 would be able to do an a-labelled transition resulting in infinitely many distinct states. We do not have a perfect solution on dealing with transitions from processes specified with unguarded recursions yet.

Moreover, the congruence property only holds on the rooted divergence-preserving branching bisimilarity. It fails for the rooted divergence-insensitive version of branching bisimilarity on TCP$^;$. Consider the following processes:

$$P_1 = \tau.\mathbf{1} \quad P_2 = (\tau.\mathbf{1})^* \quad Q = a.\mathbf{1} \ .$$

We have $P_1 \leftrightarrows_{\mathrm{rb}} P_2$ but not $P_1; Q \leftrightarrows_{\mathrm{rb}} P_2; Q$. Since the first process have an a-labelled transition followed by a τ-labelled transition, but the second process only have τ-labelled transitions.

Moreover, the standard semantics is designed to satisfy the axiom $(x + y) \cdot z = x \cdot z + y \cdot z$. However, it is no longer valid in the revised semantics. For instance, $(a + \mathbf{1}); b$ is no longer equivalent to $a; b + \mathbf{1}; b$ modulo any behavioural equivalence. In the future work, we shall provide a sound and complete axiomatisation for the process calculus TCP$^;$ with respect to strong bisimilarity as well as rooted divergence-preserving branching bisimilarity.

Another interesting future work is to establish reactive Turing powerfulness on other process calculi with non-regular iterators based on the revised semantics of the sequential composition operator. For instance, we could consider the pushdown operator "$\sharp$" and the back-and-forth operator "$\leftrightarrows$" introduced by Bergstra and Ponse in [6]. They are defined by the following equations:

$$P_1 {}^{\$} P_2 = P_1; (P_1 {}^{\$} P_2); (P_1 {}^{\$} P_2) + P_2 \quad P_1 {}^{\leftrightarrows} P_2 = P_1; (P_1 {}^{\leftrightarrows} P_2); P_2 + P_2 \ .$$

By analogy to the nesting operator, we shall also give a proper operational semantics, and then use the calculus obtained by the revised semantics to define other versions of terminating counters.

References

[1] Jos Baeten, Twan Basten & Michel Reniers (2010): *Process algebra: equational theories of communicating processes*. 50, Cambridge university press.

[2] Jos Baeten, Pieter Cuijpers, Bas Luttik & Paul van Tilburg (2009): *A process-theoretic look at automata*. In: *International Conference on Fundamentals of Software Engineering*, Springer, pp. 1–33.

[3] Jos Baeten, Pieter Cuijpers & Paul van Tilburg (2008): *A context-free process as a pushdown automaton*. In: *International Conference on Concurrency Theory*, Springer, pp. 98–113.

[4] Jos Baeten, Bas Luttik & Paul van Tilburg (2013): *Reactive Turing Machines*. *Inform. Comput.* 231, pp. 143–166, doi:10.1016/j.ic.2013.08.010.

[5] Jan Bergstra, Inge Bethke & Alban Ponse (1994): *Process algebra with iteration and nesting*. *The Computer Journal* 37(4), pp. 243–258.

[6] Jan Bergstra & Alban Ponse (2001): *Non-regular iterators in process algebra*. *Theoretical Computer Science* 269(1), pp. 203–229.

[7] Wan Fokkink, Rob van Glabbeek & Bas Luttik (2017): *Divide and Congruence III: Stability & Divergence*. In: *International Conference on Concurrency Theory*.

[8] Rob van Glabbeek (1993): *The linear time branching time spectrum II*. In: *CONCUR'93*, Springer, pp. 66–81, doi:10.1007/3-540-57208-2_6.

[9] Rob van Glabbeek, Bas Luttik & Nikola Trčka (2009): *Branching bisimilarity with explicit divergence*. *Fundamenta Informaticae* 93(4), pp. 371–392.

[10] Bas Luttik & Fei Yang (2015): *Executable Behaviour and the π-Calculus (extended abstract)*. In: *Proceedings 8th Interaction and Concurrency Experience, ICE 2015, Grenoble, France, 4-5th June 2015.*, pp. 37–52, doi:10.4204/EPTCS.189.5. Available at http://dx.doi.org/10.4204/EPTCS.189.5.

[11] Bas Luttik & Fei Yang (2016): *On the Executability of Interactive Computation*. In Arnold Beckmann, Laurent Bienvenu & Natasa Jonoska, editors: *Pursuit of the Universal - 12th Conference on Computability in Europe, CiE 2016, Paris, France, June 27 - July 1, 2016, Proceedings*, *Lecture Notes in Computer Science* 9709, Springer, pp. 312–322.

[12] Robin Milner (1989): *Communication and concurrency*. 84, Prentice hall New York etc.

[13] David Park (1981): *Concurrency and automata on infinite sequences*. In: *Theoretical computer science*, Springer, pp. 167–183.

[14] Paul van Tilburg (2011): *From computability to executability: a process-theoretic view on automata theory*.

[15] Alan Turing (1936): *On computable numbers, with an application to the Entscheidungsproblem*. *J. of Math* 58, pp. 345–363.