University of Amsterdam
Programming Research Group

Parallel Processes with Implicit Computational Capital

J.A. Bergstra
C.A. Middelburg

Report PRG0610 December 2006

J.A. Bergstra

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
The Netherlands

tel. +31 20 525.7591
e-mail: janb@science.uva.nl

C.A. Middelburg

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
The Netherlands

e-mail: keesm@win.tue.nl

Programming Research Group Electronic Report Series

Under consideration for publication in ENTCS.

Parallel Processes with Implicit Computational Capital [1]

J. A. Bergstra [a,b,2] C. A. Middelburg [a,c,3]

[a] *Programming Research Group*
University of Amsterdam
Amsterdam, the Netherlands

[b] *Department of Philosophy*
Utrecht University
Utrecht, the Netherlands

[c] *Division of Computer Science*
Eindhoven University of Technology
Eindhoven, the Netherlands

Abstract

We propose a process algebra which is concerned with processes that have an implicit computational capital. This process algebra goes along with the development that the behaviour of computer-based systems, persons and organizations is increasingly more related to money handling. It is intended to be helpful when designing systems of which the behaviour is related to money handling.

Key words: process algebra, implicit computational capital, preservation of computational money.

1 Introduction

The objective of the work reported upon in this paper is to develop new ways to use formal methods from computer science as tools for understanding money handling in a computerized setting.

Money can take different forms in practice. However, there are indications that money will become a computational phenomenon altogether: (i) the use of cash money is declining whereas different forms of electronic money are increasingly more used; (ii) in several domains, the use of cash money is already

[1] This research was carried out as part of the Jacquard-project Symbiosis, which is funded by the Netherlands Organisation for Scientific Research (NWO).

[2] Email: J.A.Bergstra@uva.nl

[3] Email: C.A.Middelburg@uva.nl

phased out in favour of forms of electronic money. An important consequence of this development is the following: it will become a matter of computational correctness that unwanted effects, such as creation of money from nowhere and leakage of money, do not take place.

It is not an easy matter to determine what are the basics of money handling. Money is what is considered and used as money by a group of people. This implies that economics, sociology and ethnography are involved in deciding what is money and what is not. For that reason, we do not aim at covering all aspects of money handling.

We propose to use the term *computational money* for quantities that are treated in a computerized setting as if they are money, ignoring the question whether from the perspective of the philosophy of economics these quantities represent money. On the one hand, thinking in terms of computational money instead of 'real money' is a drastic simplification. On the other hand, it does not preclude the possibility that computational money is at the same time considered a form of real money.

In this paper, we take up the challenge to formalize an elementary theory of computational money that is helpful when designing systems of which the behaviour is related to money handling. Rather than developing a new formalism from scratch, we will extend an existing formalism. We will add a form of computational money to ACP [3,2] to obtain a process algebra which is concerned with processes that have an implicit computational capital. This process algebra, which is called $\mathrm{ACP}_{\mathrm{icc}}$, goes along with the development that the behaviour of computer-based systems, persons and organizations is increasingly more related to money handling. We show how bisimulation models of ACP can be expanded to models of $\mathrm{ACP}_{\mathrm{icc}}$.

We also extend $\mathrm{ACP}_{\mathrm{icc}}$ with abstraction from internal actions. It happens that restrictions must be imposed on the actions from which abstraction is allowed in order to preclude unexpected effects of this kind of abstraction on the implicit computational capital of processes. The approach to abstraction from internal actions followed in this paper is based on the notion of branching bisimulation [13].

Although a subtly different approach to abstraction from internal actions is followed in CCS [9,10], the work presented in this paper can easily be adapted to CCS.

It is imaginable that, when money has become a computational phenomenon altogether, the very concept of money will be gradually superseded by new concepts that are connected with the different ways in which money handling is related to the behaviour of computer-based systems, persons and organizations. Speculating about this is interesting, but beyond the scope of computer science research.

The structure of this paper is as follows. First, we review ACP (Section 2) and guarded recursion in the setting of ACP (Section 3). Next, we introduce a simple theory about computational money values called CMV (Section 4) and

extend ACP, using CMV, to a theory about processes that have an implicit computational capital (Section 5). After that, we introduce some notions concerning transition systems which will be used later on (Section 6). Then, we show how bisimulation models of ACP can be expanded to models of $\mathrm{ACP}_{\mathrm{icc}}$ (Section 7) and discuss the notion of preservation of computational money (Section 8). Thereupon, we extend $\mathrm{ACP}_{\mathrm{icc}}$ with abstraction from internal actions (Section 9). Following this, we give examples of the use of $\mathrm{ACP}_{\mathrm{icc}}$ (Section 10). Finally, we make some concluding remarks and mention some options for future work (Section 11).

2 Algebra of Communicating Processes

In this section, we shortly review ACP (Algebra of Communicating Processes), introduced in [3]. For a comprehensive overview of ACP, the reader is referred to [2]. Although ACP is one-sorted, we make this sort explicit. The reason for this is that we will extend ACP with a second sort in Section 5.

In ACP, it is assumed that a fixed but arbitrary finite set of *actions* A, with $\delta \notin \mathsf{A}$ has been given. We write A_δ for $\mathsf{A} \cup \{\delta\}$. It is further assumed that a fixed but arbitrary commutative and associative *communication* function $| : \mathsf{A}_\delta \times \mathsf{A}_\delta \to \mathsf{A}_\delta$, such that $\delta \mid a = \delta$ for all $a \in \mathsf{A}_\delta$, has been given. The function $|$ is regarded to give the result of synchronously performing any two actions for which this is possible, and to be δ otherwise.

The algebraic theory ACP has one sort: the sort $\mathbf{P}$ of *processes*. The algebraic theory ACP has the following constants and operators:

- the *deadlock* constant $\delta : \mathbf{P}$;
- for each $a \in \mathsf{A}$, the *action* constant $a : \mathbf{P}$;
- the binary *alternative composition* operator $+ : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *sequential composition* operator $\cdot : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *parallel composition* operator $\parallel : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *left merge* operator $\between : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *communication merge* operator $| : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- for each $H \subseteq \mathsf{A}$, the unary *encapsulation* operator $\partial_H : \mathbf{P} \to \mathbf{P}$.

Terms of sorts $\mathbf{P}$ are built as usual for a one-sorted signature (see e.g. [14,11]). Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{P}$, including x, y, z.

We use infix notation for the binary operators. The following precedence conventions are used to reduce the need for parentheses. The operator $+$ binds weaker than all other binary operators to build terms of sort $\mathbf{P}$ and the operator $\cdot$ binds stronger than all other binary operators to build terms of sort $\mathbf{P}$.

Let p and q be closed terms of sort $\mathbf{P}$, $a \in \mathsf{A}$, and $H \subseteq \mathsf{A}$. Intuitively, the

Table 1
Axioms of ACP

$x + y = y + x$		A1	$x \parallel y = x \between y + y \between x + x \mid y$	CM1
$(x + y) + z = x + (y + z)$		A2	$a \between x = a \cdot x$	CM2
$x + x = x$		A3	$a \cdot x \between y = a \cdot (x \parallel y)$	CM3
$(x + y) \cdot z = x \cdot z + y \cdot z$		A4	$(x + y) \between z = x \between z + y \between z$	CM4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$		A5	$a \cdot x \mid b = (a \mid b) \cdot x$	CM5
$x + \delta = x$		A6	$a \mid b \cdot x = (a \mid b) \cdot x$	CM6
$\delta \cdot x = \delta$		A7	$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$	CM7
			$(x + y) \mid z = x \mid z + y \mid z$	CM8
			$x \mid (y + z) = x \mid y + x \mid z$	CM9
$\partial_H(a) = a$	if $a \notin H$	D1		
$\partial_H(a) = \delta$	if $a \in H$	D2	$a \mid b = b \mid a$	C1
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3	$(a \mid b) \mid c = a \mid (b \mid c)$	C2
$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$		D4	$\delta \mid a = \delta$	C3

constants and operators to build terms of sort $\mathbf{P}$ can be explained as follows:

- δ can neither perform an action nor terminate successfully;
- a first performs action a and then terminates successfully;
- $p + q$ behaves either as p or as q, but not both;
- $p \cdot q$ first behaves as p, but when p terminates successfully it continues by behaving as q;
- $p \parallel q$ behaves as the process that proceeds with p and q in parallel;
- $p \between q$ behaves the same as $p \parallel q$, except that it starts with performing an action of p;
- $p \mid q$ behaves the same as $p \parallel q$, except that it starts with performing an action of p and an action of q synchronously;
- $\partial_H(p)$ behaves the same as p, except that actions from H are blocked.

We write $\sum_{i \in \mathcal{I}} p_i$, where $\mathcal{I} = \{i_1, \ldots, i_n\}$ and $p_{i_1}, \ldots, p_{i_n}$ are terms of sort $\mathbf{P}$, for $p_{i_1} + \ldots + p_{i_n}$. The convention is that $\sum_{i \in \mathcal{I}} p_i$ stands for δ if $\mathcal{I} = \emptyset$.

The axioms of ACP are the axioms given in Table 1. CM2–CM3, CM5–CM7, C1–C3 and D1–D4 are actually axiom schemas in which a, b and c stand for arbitrary constants of sort $\mathbf{P}$ (keep in mind that also the deadlock constant belongs to the constants of sort $\mathbf{P}$) and H stands for an arbitrary subset of A.

For the main models of ACP, the reader is referred to [2].

3 Guarded Recursion

In this section, we shortly review guarded recursion in the setting of ACP.

Not all processes in a model of ACP have to be the interpretation of some closed term of sort $\mathbf{P}$. Those processes may be definable over ACP.

Table 2
Axioms for recursion

$\langle X\|E\rangle = \langle t\|E\rangle$	if $X = t \in E$	RDP
$E \Rightarrow X = \langle X\|E\rangle$	if $X \in \mathrm{V}(E)$	RSP

A process in some model of ACP is *definable* over ACP if there exists a guarded recursive specification over ACP of which that process is the unique solution.

A *recursive specification* over ACP is a set of equations $E = \{X = t_X \mid X \in V\}$ where V is a set of variables and each t_X is a term of sort $\mathbf{P}$ from the language of ACP that only contains variables from V. We write $\mathrm{V}(E)$ for the set of all variables that occur on the left-hand side of an equation in E. A *solution* of a recursive specification E is a set of processes (in some model of ACP) $\{P_X \mid X \in \mathrm{V}(E)\}$ such that the equations of E hold if, for all $X \in \mathrm{V}(E)$, X stands for P_X.

Let t be a term of sort $\mathbf{P}$ from the language of ACP containing a variable X. Then an occurrence of X in t is *guarded* if t has a subterm of the form $a \cdot t'$ where $a \in \mathsf{A}$ and t' is a term containing this occurrence of X. Let E be a recursive specification over ACP. Then E is a *guarded recursive specification* if, in each equation $X = t \in E$, all occurrences of variables in t are guarded or t can be rewritten to such a term using the axioms of ACP in either direction and/or the equations in E except the equation $X = t$ from left to right. We are only interested in models of ACP in which guarded recursive specifications have unique solutions.

For each guarded recursive specification E and each variable $X \in \mathrm{V}(E)$, we introduce a constant of sort $\mathbf{P}$ standing for the unique solution of E for X. This constant is denoted by $\langle X|E\rangle$. We often write X for $\langle X|E\rangle$ if E is clear from the context. In such cases, it should also be clear from the context that we use X as a constant.

The additional axioms for recursion are given in Table 2. In this table, we write $\langle t|E\rangle$ for t with, for all $X \in \mathrm{V}(E)$, all occurrences of X in t replaced by $\langle X|E\rangle$. Both RDP and RSP are axiom schemas. Side conditions are added to restrict the variables, terms and guarded recursive specifications for which X, t and E stand.

We will write ACP+REC for ACP extended with the constants standing for the unique solutions of guarded recursive specifications and the axioms RDP and RSP.

Each closed term of sort $\mathbf{P}$ from the language of ACP denotes a finite process, i.e. a process of which the length of the sequences of actions that it can perform is bounded. However, not each closed term over the signature of ACP+REC denotes a finite process: recursion gives rise to infinite processes. Closed terms over the signature of ACP+REC that denote the same infinite process cannot always be proved equal by means of the axioms of ACP+REC. To remedy this, we introduce the approximation induction principle.

Table 3
Approximation induction principle

$\bigwedge_{n\geq 0}\pi_n(x) = \pi_n(y) \Rightarrow x = y$	AIP

Table 4
Axioms for projection operators

$\pi_0(a) = \delta$	PR1
$\pi_{n+1}(a) = a$	PR2
$\pi_0(a \cdot x) = \delta$	PR3
$\pi_{n+1}(a \cdot x) = a \cdot \pi_n(x)$	PR4
$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$	PR5

The approximation induction principle, AIP in short, is based on the view that two processes are identical if their approximations up to any finite depth are identical. The approximation up to depth n of a process behaves the same as that process, except that it cannot perform any further action after n actions have been performed.

AIP is the infinitary conditional equation given in Table 3. Here, approximation up to depth n is phrased in terms of a unary *projection* operator π_n. The axioms for the projection operators are given in Table 4. Axioms PR1–PR5 are actually axiom schemas in which a stands for an arbitrary constants of sort $\mathbf{P}$ and n stands for an arbitrary natural number.

Let T stand for either ACP or ACP+REC. Then we will write T+PR for T extended with the projections operators π_n and the axioms PR1–PR5.

AIP is consistent with ACP+PR and ACP+REC+PR, but it does not hold in all models for ACP and ACP+REC. RSP is derivable from the axioms of ACP, RDP and AIP.

4 Computational Money Values

In this section, we present an algebraic theory about computational money values. The presented theory is called CMV.

The algebraic theory CMV has one sort: the sort $\mathbf{M}$ of *computational money values*. The algebraic theory CMV has the following constants and operators:

- the constant $0 : \mathbf{M}$;
- the constant $1 : \mathbf{M}$;
- the binary *addition* operator $+ : \mathbf{M} \times \mathbf{M} \to \mathbf{M}$;
- the unary *additive inverse* operator $- : \mathbf{M} \to \mathbf{M}$;
- the binary *multiplication* operator $\cdot : \mathbf{M} \times \mathbf{M} \to \mathbf{M}$;
- the binary *maximum* operator $\max : \mathbf{M} \times \mathbf{M} \to \mathbf{M}$.

Terms of sort $\mathbf{M}$ are built as usual for a one-sorted signature. Throughout the

Table 5
Axioms for computational money values

$(i + j) + k = i + (j + k)$	CMV1
$i + j = j + i$	CMV2
$i + 0 = i$	CMV3
$i + -i = 0$	CMV4
$(i \cdot j) \cdot k = i \cdot (j \cdot k)$	CMV5
$i \cdot j = j \cdot i$	CMV6
$i \cdot 1 = i$	CMV7
$i \cdot (j + k) = i \cdot j + i \cdot k$	CMV8
$\max(i, j) = \max(j, i)$	CMV9
$\max(i + k, j + k) = \max(i, j) + k$	CMV10
$\max(0, i^2 + j^2 + k^2 + l^2) = i^2 + j^2 + k^2 + l^2$	CMV11

paper, we assume that there are infinitely many variables of sort $\mathbf{M}$, including i, j, k, l.

As usual, we use prefix notation for the unary operator $-$ and infix notation for the binary operators $+$ and $\cdot$. The following additional precedence convention is used to reduce the need for parentheses. The binary operator $+$ binds weaker than all other operators to build terms of sort $\mathbf{M}$ and the unary operator $-$ binds stronger than all other operators to build terms of sort $\mathbf{M}$. We introduce the following abbreviation: m^2, where m is term of sort $\mathbf{M}$, abbreviates $m \cdot m$.

The constants and operators of CMV are adopted from integer arithmetic and need no further explanation.

The axioms of CMV are the axioms given in Tables 5. Axioms CMV1–CMV8 are the axioms of a commutative ring with unit. Axioms CMV9–CMV11 are the defining axioms of max. In axiom CMV11, use is made of Lagrange's theorem that every natural number can be represented as the sum of four squares.

The initial model of CMV is considered the standard model of CMV.

5 Parallel Processes with Computational Capital

In this section, we take up the extension of ACP to a theory about processes that have an implicit computational capital. The result is called $\mathrm{ACP}_{\mathrm{icc}}$.

First of all, we give a picture of the ideas that underlie the design of this extension. There are three kinds of actions: actions with which an amount of computational money is spent, actions with which an amount of computational money is acquired and actions with which no computational money is spent or acquired. As for spending or acquiring computational money, the effect of performing two actions synchronously is the joint effect of the two actions concerned.

The implicit computational capital of a process is the amount of compu-

tational money that the process has available to spend by performing actions, measured as a natural number. As the name suggests, the implicit computational capital of a process is looked upon as an implicit property of the process. It is the least amount of computational money that is needed to account for the behaviour of the process.[4] Neither what a process is capable of using exceeds its implicit computational capital nor the other way round. The fact that the implicit computational capital of a process does not exceed what the process is capable of using corresponds to the liquidity assumption that the whole capital can potentially be used. The monetary effects of performing actions do not affect the behaviour of a process that has an implicit computational capital.

Now for the presentation of the extension of $\mathrm{ACP}_{\mathrm{icc}}$. It has the sorts, constants and operators of both ACP and CMV, the projection operators π_n, and in addition the following constants and operators to build terms of sort $\mathbf{M}$:

- for each $a \in \mathsf{A}$, the *computational money transfer* constant $\mathrm{q}(a) : \mathbf{M}$;
- the unary *implicit computational capital* operator $\mathrm{Q} : \mathbf{P} \to \mathbf{M}$.

The projection operators are included because they are needed in the axioms concerning the implicit computational capital of infinite processes.

Let p be a closed term of sort $\mathbf{P}$. Intuitively, the computational money transfer constants and the implicit computational capital operator can be explained as follows:

- if $\mathrm{q}(a) > 0$, then $\mathrm{q}(a)$ is the amount of computational money spent by a process on performing action a;
- if $\mathrm{q}(a) < 0$, then $-\mathrm{q}(a)$ is the amount of computational money acquired by a process on performing action a;
- if $\mathrm{q}(a) = 0$, then nothing is spent or acquired by a process on performing action a;
- $\mathrm{Q}(p)$ is the amount of computational money that process p has available to spend by performing actions.

If $\mathrm{q}(a) = 0$, then action a is called *CM-neutral*.

It is assumed that for each $a \in \mathsf{A}$, a fixed but arbitrary equation $\mathrm{q}(a) = m$, where m is a closed term of sort $\mathbf{M}$ from the language of CMV has been given. We write CMT for the set of all those equations. It is further assumed that, for all $a, b, c \in \mathsf{A}$ for which $a \,|\, b = c$, the equation $\mathrm{q}(c) = \mathrm{q}(a) + \mathrm{q}(b)$ is derivable from CMT and the axioms of CMV.

The axioms of $\mathrm{ACP}_{\mathrm{icc}}$ are the axioms of ACP, the axioms of CMV, axioms PR1–PR5 from Table 4, the equations from CMT, and the axioms given in

[4] In ACP-like process algebras, it is usual to consider the terms process and behaviour synonyms. This is not the case in $\mathrm{ACP}_{\mathrm{icc}}$, where a process is made up of behaviour and implicit computational capital.

Table 6
Axioms for computational money transfer

$\mathrm{q}(c) = \mathrm{q}(a) + \mathrm{q}(b) \quad \mathsf{if}\ a \mid b = c$	CMTC

Table 7
Axioms for implicit computational capital

$\mathrm{Q}(\delta) = 0$	ICC1
$\mathrm{Q}(a) = \max(\mathrm{q}(a), 0)$	ICC2
$\max(\mathrm{Q}(x), 0) = \mathrm{Q}(x) \Rightarrow \mathrm{Q}(a \cdot x) = \max(\mathrm{q}(a) + \mathrm{Q}(x), 0)$	ICC3
$\max(\mathrm{Q}(x), 0) = \mathrm{Q}(x) \wedge \max(\mathrm{Q}(y), 0) = \mathrm{Q}(y) \Rightarrow \mathrm{Q}(x + y) = \max(\mathrm{Q}(x), \mathrm{Q}(y))$	ICC4
$\bigwedge_{m \geq 0} \max(\mathrm{Q}(\pi_m(x)), \mathrm{Q}(\pi_n(x))) = \mathrm{Q}(\pi_n(x)) \Rightarrow \mathrm{Q}(x) = \mathrm{Q}(\pi_n(x))$	ICC5
$\bigwedge_{n \geq 0} \bigvee_{m \geq 0} \max(\mathrm{Q}(\pi_m(x)), \mathrm{Q}(\pi_n(x)) + 1) = \mathrm{Q}(\pi_m(x)) \Rightarrow \mathrm{Q}(x) = -1$	ICC6

Tables 6 and 7. CMTC and ICC2–ICC3 are actually axiom schemas in which a, b and c stand for arbitrary constants of sort $\mathbf{P}$ different from δ.

The equations from CMT and axiom CMTC concern spending and acquiring computational money on performing actions. The equations from CMT are the defining equations of the computational money transfer constants. CMTC expresses that, as for spending or acquiring computational money, the effect of performing two actions synchronously is the joint effect of the two actions concerned.

The implicit computational capital of a process is the amount of computational money that the process is capable of using. This amount cannot be negative. However, it can be undefined if the process is infinite. In order to circumvent the use of algebras with partial operations, which is not considered in elementary algebraic specifications (see e.g. [7]), -1 is used to represent the undefinedness of the implicit computational capital of a process. Axioms ICC1–ICC4 cover the case where the process is finite. Axioms ICC5 and ICC6 cover the case where the process is infinite. Axiom ICC5 states that the implicit computational capital of a process is the greatest capital among the implicit computational capitals of all finite approximations of the process if it exists. Otherwise, axiom ICC6 applies. ICC5 and ICC6 are infinitary formulas, but ICC5 is still a conditional equation.

Proposition 5.1 *For all closed terms p_1 and p_2 of sort $\mathbf{P}$ from the language of* ACP*, the following is derivable from the axioms of* $\mathrm{ACP}_{\mathrm{icc}}$*:*

$$\mathrm{Q}(p_1 \cdot p_2) \leq \mathrm{Q}(p_1) + \mathrm{Q}(p_2)\ , \qquad \mathrm{Q}(p_1 \parallel p_2) = \mathrm{Q}(p_1) + \mathrm{Q}(p_2)\ ,$$
$$\mathrm{Q}(p_1 + p_2) \leq \mathrm{Q}(p_1) + \mathrm{Q}(p_2)\ , \qquad \mathrm{Q}(p_1 \between p_2) \leq \mathrm{Q}(p_1) + \mathrm{Q}(p_2)\ ,$$
$$\mathrm{Q}(\partial_H(p_1)) \leq \mathrm{Q}(p_1)\ , \qquad \mathrm{Q}(p_1 \mid p_2) \leq \mathrm{Q}(p_1) + \mathrm{Q}(p_2)\ .$$

Proof. Straightforward, by induction on the structure p_1 and p_2. □

Guarded recursion can be added to $\mathrm{ACP}_{\mathrm{icc}}$ as it is added to ACP in Section 3. We will write $\mathrm{ACP}_{\mathrm{icc}}$+REC for the resulting theory.

The following example goes into the implicit computational capital of an

infinite process. In case $\mathrm{q}(a) > 0$, the process $\langle X|\{X = a \cdot X\}\rangle$ repeats forever a step on which the amount $\mathrm{q}(a)$ is spent. Because no capital is large enough for that, the implicit computational capital of $\langle X|\{X = a \cdot X\}\rangle$ is undefined if $\mathrm{q}(a) > 0$. Assuming $\mathrm{q}(a) > 0$, we derive from axioms RDP, PR3, PR4 and ICC6, using an inductive argument, that $\mathrm{Q}(\langle X|\{X = a \cdot X\}\rangle) = -1$. In case $\mathrm{q}(a) < 0$, the process $\langle X|\{X = a \cdot X\}\rangle$ repeats forever a step on which the amount $-\mathrm{q}(a)$ is acquired. In case $\mathrm{q}(a) = 0$, the process $\langle X|\{X = a \cdot X\}\rangle$ repeats forever a step on which nothing is spent or acquired. Because in both cases no capital is needed, the implicit computational capital of $\langle X|\{X = a \cdot X\}\rangle$ is zero if $\mathrm{q}(a) \leq 0$. Assuming $\mathrm{q}(a) \leq 0$, we derive from axioms RDP, PR3, PR4 and ICC5, using an inductive argument, that $\mathrm{Q}(\langle X|\{X = a \cdot X\}\rangle) = 0$.

Let p be a closed term of sort $\mathbf{P}$. Then p denotes a "money source" iff $\mathrm{Q}(p) = -1$. It is not that easy to characterize "money sinks", i.e. processes that accumulate computational money, in terms of Q. Exclusion of money sources by taking the infinitary formula

$$\bigvee_{n \geq 0} \bigwedge_{m \geq 0} \max(\mathrm{Q}(\pi_m(x)), \mathrm{Q}(\pi_n(x))) = \mathrm{Q}(\pi_n(x))$$

as additional axiom is possible only if we drop RDP: it is inconsistent with RDP.

The set of formulas represented by the schema

$$\mathrm{q}(a) \neq 0 \Rightarrow \bigvee_{a' \in \mathsf{A}} (a \mid a' \neq \delta \ \wedge \ \mathrm{q}(a) + \mathrm{q}(a') = 0) \ ,$$

where a stands for an arbitrary constant of sort $\mathbf{P}$ different from δ, can be added to the axioms of $\mathrm{ACP}_{\mathrm{icc}}$. These formulas express that, for each action with which an amount of computational money is spent, there exists another action with which it can be performed synchronously and the same amount is acquired, and similarly, for each action with which an amount of computational money is acquired, there exists another action with which it can be performed synchronously and the same amount is spent. In other words, they say that the monetary effect of performing an action can always be neutralized by performing it synchronously with some other action. It can be useful to hypothesize this neutralization property in derivations concerning a system that is not supposed to interact with its environment, but not otherwise.

In $\mathrm{ACP}_{\mathrm{icc}}$, amounts of computational money are not used for book-keeping of debts. The reason for this is that implicitly abstracting from the process to which a debt is owed is far from obvious.

The models of $\mathrm{ACP}_{\mathrm{icc}}$ of which the restriction to the signature of CMV is the initial model of CMV are considered the standard models of $\mathrm{ACP}_{\mathrm{icc}}$.

6 Transition Systems Induced by Models of ACP

In this section, we introduce the notions of transition system induced by a model of ACP, paths in a transition system and bisimilarity of transition

systems. Those notions will be used in Section 7 to show how bisimulation models of ACP can be expanded to models of $\mathrm{ACP}_{\mathrm{icc}}$. Prior to all that, we make precise what we understand by a transition system.

A *transition system* consists of the following:

- a set S of *states*;
- a set $\xrightarrow{a} \subseteq S \times S$, for each $a \in \mathsf{A}$;
- a set $\xrightarrow{a}\surd \subseteq S$, for each $a \in \mathsf{A}$;
- an *initial state* $s^0 \in S$.

We write $s \xrightarrow{a} s'$ instead of $(s, s') \in \xrightarrow{a}$ and $s \xrightarrow{a} \surd$ instead of $s \in \xrightarrow{a}\surd$. Furthermore, we write $\rightarrow$ for the family of sets $(\xrightarrow{a})_{a\in\mathsf{A}}$ and $\rightarrow\surd$ for the family of sets $(\xrightarrow{a}\surd)_{a\in\mathsf{A}}$.

Let $T_1 = (S_1, \rightarrow_1, \rightarrow\surd_1, s^0_1)$ and $T_2 = (S_2, \rightarrow_2, \rightarrow\surd_2, s^0_2)$ be transition systems. Then a *bisimulation* B between T_1 and T_2 is a binary relation $B \subseteq S_1 \times S_2$ such that $B(s^0_1, s^0_2)$ and for all s_1, s_2 such that $B(s_1, s_2)$:

- $s_1 \xrightarrow{a}\surd_1$ iff $s_2 \xrightarrow{a}\surd_2$;
- if $s_1 \xrightarrow{a}_1 s'_1$, then there is a state s'_2 such that $s_2 \xrightarrow{a}_2 s'_2$ and $B(s'_1, s'_2)$;
- if $s_2 \xrightarrow{a}_2 s'_2$, then there is a state s'_1 such that $s_1 \xrightarrow{a}_1 s'_1$ and $B(s'_1, s'_2)$.

Two transition systems T_1 and T_2 are *bisimilar*, written $T_1 \leftrightarroweq T_2$, if there exists a bisimulation B between T_1 and T_2.

Let $T = (S, \rightarrow, \rightarrow\surd, s^0)$ be a transition system. Then the set of *paths* in T, written $\mathrm{P}(T)$, is the smallest subset of $\{\pi \frown \langle s\rangle \mid \pi \in (S \times \mathsf{A})^* \wedge s \in S\}$ such that: [5]

- $\langle s^0\rangle \in \mathrm{P}(T)$,
- if $\pi \frown \langle s\rangle \in \mathrm{P}(T)$ and $s \xrightarrow{a} s'$, then $\pi \frown \langle s, a, s'\rangle \in \mathrm{P}(T)$.

A transition system may have states that are not reachable from its initial state by a path in the transition system. We exclude transition systems with unreachable states as follows.

Let $T = (S, \rightarrow, \rightarrow\surd, s^0)$ be a transition system. The *connected part* of T, written $\Gamma(T)$, is defined as follows:

$$\Gamma(T) = (S', \rightarrow', \rightarrow\surd', s^0) \,,$$

where

$$S' = \{s \in S \mid \exists \pi \in (S \times \mathsf{A})^* \bullet \pi \frown \langle s\rangle \in \mathrm{P}(T)\} \,,$$

and for every $a \in \mathsf{A}$:

$$\begin{aligned} \xrightarrow{a}' &= \xrightarrow{a} \cap (S' \times S') \,, \\ \xrightarrow{a}\surd' &= \xrightarrow{a}\surd \cap S' \,. \end{aligned}$$

[5] We write $\langle\,\rangle$ for the empty sequence, $\langle e\rangle$ for the sequence having e as sole element and $\sigma \frown \sigma'$ for the concatenation of sequences σ and σ'; and we use $\langle e_1, \ldots, e_n\rangle$ as a shorthand for $\langle e_1\rangle \frown \ldots \frown \langle e_n\rangle$.

Let $\mathfrak{A}$ be a model of ACP, let P be the domain of $\mathfrak{A}$, and let $p \in P$. Then the transition system of p *induced by* $\mathfrak{A}$, written $\mathrm{TS}(\mathfrak{A}, p)$, is the transition system $\Gamma(P, \rightarrow, \rightarrow\surd, p)$ where, for every $a \in \mathsf{A}$:

$$\begin{aligned} \xrightarrow{a} &= \{(p_1, p_2) \mid \mathfrak{A} \models x_1 = x_1 + a \cdot x_2\ [p_1, p_2]\}\ , \\ \xrightarrow{a}\surd &= \{p_1 \mid \mathfrak{A} \models x = x + a\ [p_1]\}\ . \end{aligned}$$

7 Expanding Models of ACP to Models of $\mathrm{ACP}_{\mathrm{icc}}$

In $\mathrm{ACP}_{\mathrm{icc}}$, the implicit computational capital of a process is the least amount of computational money that is needed to account for the behaviour of the process. This amount is implicit in the behaviour of the process. In this section, we show that each bisimulation model of ACP can be expanded to a model of $\mathrm{ACP}_{\mathrm{icc}}$, provided that it can be expanded to a model of ACP+PR.

Let $\mathfrak{A}$ be a model of ACP. Then $\mathfrak{A}$ is a *bisimulation model* of ACP if for all p and p' from the domain of $\mathfrak{A}$, $\mathrm{TS}(\mathfrak{A}, p) \leftrightarroweq \mathrm{TS}(\mathfrak{A}, p')$ implies $p = p'$.

In [6], the full bisimulation models of $\mathrm{ACP}^{\mathrm{fo}}$, a first-order extension of ACP, are introduced. The full bisimulation models of ACP are the restrictions of those models to the signature of ACP. The full bisimulation models of ACP are the main models of ACP. We gather from [6]:

- the full bisimulation models of ACP are bisimulation models;
- each bisimulation model of ACP can be isomorphically embedded in one of the full bisimulation models of ACP;
- in each full bisimulation model of ACP, all finite and countably infinite guarded recursive specifications over ACP have a unique solution;
- all full bisimulation models of ACP can be expanded to models of ACP+PR;
- the full bisimulation model of ACP with the smallest domain can be expanded to a model of ACP+PR in which AIP holds.

Below, we will write $\mathrm{ACP}_{\mathrm{cm}}$ for $\mathrm{ACP}_{\mathrm{icc}}$ without the implicit computational capital operator Q and consequently without the axioms ICC1–ICC6. Moreover, we write CMV+CMT for CMV extended with the computational money transfer constants $\mathrm{q}(a)$ and the equations from CMT.

Lemma 7.1 *For each bisimulation model* $\mathfrak{A}$ *of* ACP *that can be expanded to a model of* ACP+PR *and each model* $\mathfrak{B}$ *of* CMV*, there exists a model* $\mathfrak{C}$ *of* $\mathrm{ACP}_{\mathrm{cm}}$ *such that the restriction of* $\mathfrak{C}$ *to the signature of* ACP *is* $\mathfrak{A}$ *and the restriction of* $\mathfrak{C}$ *to the signature of* CMV *is* $\mathfrak{B}$*.*

Proof. Let $\mathfrak{A}$ be a bisimulation model of ACP that can be expanded to a model of ACP+PR, and let $\mathfrak{A}'$ be an expansion of $\mathfrak{A}$ to a model of ACP+PR. Let $\mathfrak{B}$ be a model of CMV, and let $\mathfrak{B}'$ be the unique expansion of $\mathfrak{B}$ to a model of CMV+CMT. This unique expansion exists because CMV+CMT is a definitional extension of CMV. Moreover, the instances of axiom schema CMTC hold in $\mathfrak{B}'$ because they are derivable from CMT and the axioms of

CMV. From this and the fact that the signatures of $\mathfrak{A}'$ and $\mathfrak{B}'$ are disjoint, it follows, by the amalgamation result about expansions presented as Theorem 6.1.1 in [8] (adapted to the many-sorted case), that there exists a model $\mathfrak{C}$ of $\mathrm{ACP}_{\mathrm{cm}}$ such that the restriction to the signature of ACP+PR is $\mathfrak{A}'$ and the restriction to the signature of CMV+CMT is $\mathfrak{B}'$. Because $\mathfrak{A}'$ is an expansion of $\mathfrak{A}$ and $\mathfrak{B}'$ is an expansion of $\mathfrak{B}$, the restriction of $\mathfrak{C}$ to the signature of ACP is $\mathfrak{A}$ and the restriction of $\mathfrak{C}$ to the signature of CMV is $\mathfrak{B}$. □

Theorem 7.2 *For each bisimulation model $\mathfrak{A}$ of* ACP *that can be expanded to a model of* ACP+PR*, there exists a model $\mathfrak{A}'$ of $\mathrm{ACP}_{\mathrm{icc}}$ such that the restriction of $\mathfrak{A}'$ to the signature of* ACP *is $\mathfrak{A}$.*

Proof. Let $\mathfrak{A}$ be a bisimulation model of ACP that can be expanded to a model of ACP+PR. Let $\mathfrak{B}$ be a model of CMV, and let $\leq$ be the ordering on the domain of $\mathfrak{B}$ such that, for all i and j from the domain of $\mathfrak{B}$, $i \leq j$ iff $\max(i, j) = j$. It is easy to see that the subset of the domain of $\mathfrak{B}$ that consists of all elements that are the interpretation of some closed term of sort $\mathbf{M}$ is totally ordered by $\leq$. By Lemma 7.1, there exists a model $\mathfrak{C}$ of $\mathrm{ACP}_{\mathrm{cm}}$ of which the restriction to the signature of ACP is $\mathfrak{A}$ and the restriction to the signature of CMV is $\mathfrak{B}$. Let $\mathfrak{A}'$ be $\mathfrak{C}$ expanded with the function Q from the domain of $\mathfrak{A}$ to the domain of $\mathfrak{B}$ such that, for all p from the domain of $\mathfrak{A}$:

- $\mathrm{Q}(p)$ is the greatest element of $\{\mathrm{Q}'(\pi) \mid \pi \in \mathrm{P}(\mathrm{TS}(\mathfrak{A}, p))\}$ with respect to $\leq$, where Q' is defined inductively by $\mathrm{Q}'(\langle p' \rangle) = 0$ and $\mathrm{Q}'(\langle p', a \rangle \frown \pi) = \max(\mathrm{q}(a) + \mathrm{Q}'(\pi), 0)$, if it exists;
- $\mathrm{Q}(p)$ is -1 otherwise.

Because $\mathfrak{C}$ is an expansion of $\mathfrak{A}$, the restriction of $\mathfrak{A}'$ to the signature of ACP is $\mathfrak{A}$. It remains to be proved that $\mathfrak{A}'$ is a model of $\mathrm{ACP}_{\mathrm{icc}}$. Because $\mathfrak{A}'$ is an expansion of $\mathfrak{C}$, it is sufficient to prove that ICC1–ICC6 hold in $\mathfrak{A}'$. Moreover, because each bisimulation model of ACP can be isomorphically embedded in one of the full bisimulation models of ACP, it may be assumed that $\mathfrak{A}$ is a full bisimulation model of ACP. Then $\mathfrak{A}$ can be expanded to a model of ACP+PR in a similar way as graph models of ACP are expanded with projection operations in [2]. Without loss of generality, it may be assumed that the restriction of $\mathfrak{C}$ to the signature of ACP+PR is the expansion of $\mathfrak{A}$ to a model of ACP+PR obtained in that way. Let $\mathfrak{A}''$ be this expansion of $\mathfrak{A}$. Then for all p from the domain of $\mathfrak{A}$, $\mathrm{P}(\mathrm{TS}(\mathfrak{A}'', \pi_n(p))) = \{\pi \in \mathrm{P}(\mathrm{TS}(\mathfrak{A}'', p)) \mid len(\pi) \leq n\}$, where $len(\pi)$ is the length of the sequence from A^* obtained by leaving out all states in π. From this and the construction of the full bisimulation models of ACP given in [6], it follows easily that ICC1–ICC6 hold in $\mathfrak{A}'$. □

8 Preservation of Computational Money

In this section, we discuss the notion of preservation of computational money.

Let $\mathfrak{A}$ be a model of $\mathrm{ACP}_{\mathrm{icc}}$, let P be the domain associated with $\mathbf{P}$ in $\mathfrak{A}$, let $p \in P$, and let $T = (P', \to, \to\surd, p)$ be the transition system of p induced by $\mathfrak{A}$. Then p is *CM-preserving* if

- $\mathrm{Q}(p') = \mathrm{q}(a)$ for all $p' \in P'$ and $a \in \mathsf{A}$ such that $p' \xrightarrow{a} \surd$;
- $\mathrm{Q}(p') = \mathrm{q}(a) + \mathrm{Q}(p'')$ for all $p', p'' \in P'$ and $a \in \mathsf{A}$ such that $p' \xrightarrow{a} p''$;
- $\mathrm{Q}(p') = \mathrm{Q}(p'')$ for all $p', p'' \in P \setminus \{\delta\}$ such that $p' + p'' \in P'$.

Using the terminology of [6], CM-preserving belongs to the external properties of processes in models of $\mathrm{ACP}_{\mathrm{icc}}$. It is an open problem whether a first-order axiomatization of this property is possible.

The next proposition states that the implicit computational capital of a CM-preserving process remains constant if it evolves through a number of CM-neutral steps. To make precise what it means to evolve through a number of CM-neutral steps, we introduce the notion of CM-neutral paths.

Let $T = (S, \to, \to\surd, s^0)$ be a transition system. Then the set of *CM-neutral paths* in T, written $\mathrm{NP}(T)$, is the smallest subset of $\mathrm{P}(T)$ such that:

- $\langle s^0 \rangle \in \mathrm{NP}(T)$;
- if $\pi \frown \langle s \rangle \in \mathrm{NP}(T)$, $s \xrightarrow{a} s'$ and $\mathrm{q}(a) = 0$, then $\pi \frown \langle s, a, s' \rangle \in \mathrm{NP}(T)$.

Proposition 8.1 *Let $\mathfrak{A}$ be a model of $\mathrm{ACP}_{\mathrm{icc}}$, let P be the domain associated with $\mathbf{P}$ in $\mathfrak{A}$, let $p \in P$, let $P' \subseteq P$ be the set of states of $\mathrm{TS}(\mathfrak{A}, p)$, and let $p' \in P'$. Then, for all $\pi \in (P' \times \mathsf{A})^*$, p is CM-preserving and $\pi \frown \langle p' \rangle \in \mathrm{NP}(\mathrm{TS}(\mathfrak{A}, p))$ implies $\mathrm{Q}(p) = \mathrm{Q}(p')$.*

Proof. Straightforward, by induction on the length of π. □

It is plausible that an electronic device, such as an electronic wallet, is modelled in $\mathrm{ACP}_{\mathrm{icc}}$ as a process that deadlocks if the device crashes because of a technical failure. In such cases, it is likely that the process is not CM-preserving. This indicates that preservation of computational money is not a fact of life, but a constraint on how to model. Modelling with CM-preserving processes may enforce the introduction of a hypothetical process to which computational money is transferred just before the crash takes place. The problem with that process is that, unless it is able to dispose of its computational money, it accumulates computational money and by doing so it is not CM-preserving.

In general, in order to make a process that accumulates computational money CM-preserving, it must be made capable of sending that computational money via a ghost channel that is only used for sending.

As an example, we consider the process $\langle X | \{X = a \cdot X\} \rangle$ where $\mathrm{q}(a) = -1$. This process is clearly accumulating computational money and therefore not CM-preserving. We assume that for all closed terms m and m' of sort $\mathbf{M}$ from

the language of CMV with $\max(m,1) = m$ and $\max(m',1) = m'$, respectively: $s_{ghost}(m) \in \mathsf{A}$, $s_{ghost}(m) \mid s_{ghost}(m') = s_{ghost}(m+m')$ and $\mathrm{q}(s_{ghost}(m)) = m$. The process $\langle X|\{X = a \cdot (X \parallel s_{ghost}(1))\}\rangle$ behaves as $\langle X|\{X = a \cdot X\}\rangle$, except that it can send its computational money in arbitrary amounts at once when needed.

9 Abstraction from Internal Actions

In this section, we extend $\mathrm{ACP}_{\mathrm{icc}}$ with abstraction from internal actions. The result is called $\mathrm{ACP}^{\tau}_{\mathrm{icc}}$. In [4], abstraction from internal actions was first introduced in ACP. The approach to abstraction from internal actions followed in that paper is based on the notion of observation equivalence which originates from [9]. In this paper, a subtly different approach is followed. It is based on the notion of branching bisimulation equivalence which originates from [12].

The sorts, constants and operators of $\mathrm{ACP}^{\tau}_{\mathrm{icc}}$ are those of $\mathrm{ACP}_{\mathrm{icc}}$, and in addition:

- the *silent step* constant $\tau : \mathbf{P}$;
- for each $I \subseteq \{a \in \mathsf{A} \mid \mathrm{q}(a) = 0\}$, the unary *abstraction* operator $\tau_I : \mathbf{P} \to \mathbf{P}$.

Let p be a closed term of sort $\mathbf{P}$, and $I \subseteq \{a \in \mathsf{A} \mid \mathrm{q}(a) = 0\}$. Intuitively, the silent step constant and the abstraction operators can be explained as follows:

- τ performs an action that is unobservable and then terminates successfully;
- $\tau_I(p)$ behaves the same as as p, except that actions from I are turned into silent steps.

It is assumed that $\tau \notin \mathsf{A}_\delta$. We write $\mathsf{A}_{\delta\tau}$ for $\mathsf{A}_\delta \cup \{\tau\}$. The communication function $\mid : \mathsf{A}_\delta \times \mathsf{A}_\delta \to \mathsf{A}_\delta$ is extended to a function $\mid : \mathsf{A}_{\delta\tau} \times \mathsf{A}_{\delta\tau} \to \mathsf{A}_\delta$ by stipulating that $a \mid \tau = \tau \mid a = \delta$ for all $a \in \mathsf{A}_{\delta\tau}$.

The axioms of $\mathrm{ACP}^{\tau}_{\mathrm{icc}}$ are the axioms of $\mathrm{ACP}_{\mathrm{icc}}$ and the axioms given in Table 8. Axioms TI1–TI4 and C4 are actually axiom schemas in which a stands for an arbitrary constant of sort $\mathbf{P}$ and I stands for an arbitrary subset of $\{a \in \mathsf{A} \mid \mathrm{q}(a) = 0\}$.

Axioms B1 and B2 reflect the intuition that the presence of a non-initial silent step cannot be inferred from the observable behaviour of a process if the process can perform that silent step without discarding any of the alternatives that it had before. Axioms TI1–TI4 says that abstraction from certain actions turns those actions into silent steps. Axioms PRT1 and PRT2 express that the silent step does not count in projections. Abstraction from actions that are not CM-neutral is excluded because it is considered counter-intuitive to take actions that are not CM-neutral for internal. In agreement with that, axiom CMTT states that the silent step is CM-neutral. Notice that by this axiom abstraction from actions that are not CM-neutral could lead to change of implicit computational capital as side-effect.

Table 8
Axioms for abstraction

$x \cdot \tau = x$		B1
$x \cdot (\tau \cdot (y + z) + y) = x \cdot (y + z)$		B2
$\tau_I(a) = a$	if $a \notin I$	TI1
$\tau_I(a) = \tau$	if $a \in I$	TI2
$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$		TI3
$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$		TI4
$a \mid \tau = \delta$		C4
$\pi_n(\tau) = \tau$		PRT1
$\pi_n(\tau \cdot x) = \tau \cdot \pi_n(x)$		PRT2
$\mathrm{q}(\tau) = 0$		CMTT

Guarded recursion can be added to $\mathrm{ACP}^{\tau}_{\mathrm{icc}}$ as it is added to ACP in Section 3. The definition of guardedness of an occurrence of a variable in a term must be taken strictly. For example, the rightmost occurrence of X in $a \cdot X + \tau \cdot X$ is not guarded.

ACP extended with the silent step constant τ, the abstraction operator τ_I for each $I \in \mathsf{A}$, and the axioms B1, B2 and TI1–TI4, is known as ACP^{τ}. Notice that, different from $\mathrm{ACP}^{\tau}_{\mathrm{icc}}$, ACP^{τ} has an abstraction operator for each $I \subseteq \mathsf{A}$. For a comprehensive overview of ACP^{τ}, the reader is referred to [2].

10 Some Examples

In this section, we illustrate the use of $\mathrm{ACP}_{\mathrm{icc}}$ by means of some examples concerning simple vending machines. We have aimed at examples that make an aid in understanding of the notion of implicit computational capital of a process.

If it is possible to perform two actions synchronously, then it is usual that the nature of one of them is active and the nature of the other is passive. In such cases, we use the convention that the constant for the active action with the prefix $\sim$ is taken as the constant for the passive action. Moreover, the constant of the active action with overlining is taken as the constant for the action that is the result of synchronously performing the active action and the passive action. We take the $\mid : \mathsf{A}_\delta \times \mathsf{A}_\delta \to \mathsf{A}_\delta$ such that for all $a \in \mathsf{A}$ for which $\sim a \in \mathsf{A}$:

- $a \mid \sim a = \overline{a}$;
- $a \mid b = \delta$ for all $b \in \mathsf{A} \setminus \{\sim a\}$;
- $b \mid \sim a = \delta$ for all $b \in \mathsf{A} \setminus \{a\}$.

For all $a \in \mathsf{A}$ for which $\sim a \in \mathsf{A}$, we will always take $\mathrm{q}(a)$ and $\mathrm{q}(\sim a)$ such that $\mathrm{q}(a) + \mathrm{q}(\sim a) = 0$.

10.1 Example 1

The first example concerns a very simple vending machine. A cup of coffee is the only product that can be bought. Two additives can be ordered, namely milk and sugar. The price of a cup of coffee is €0.50. Payments can be made by means of coins with value €0.50 only. The coins are collected in a cash-box with a capacity of 400 coins. The cash-box can be emptied at any time. This simple vending machine, with k_0 coins in the cash-box ($k_0 \in \{0, \ldots, 400\}$), can be defined by the guarded recursive specification over $\mathrm{ACP}_{\mathrm{icc}}$ that consists of the following equations:

$$\begin{aligned}
VM1_{k_0} &= VM1'_{\emptyset,k_0}\\
VM1'_{as,k} &= {\sim}push_button(milk) \cdot VM1'_{as\cup\{milk\},k}\\
&\quad + {\sim}push_button(sugar) \cdot VM1'_{as\cup\{sugar\},k}\\
&\quad + {\sim}insert_coin \cdot deliver_coffee(as) \cdot VM1'_{\emptyset,k+1}\\
&\quad + {\sim}empty_cash_box(k) \cdot VM1'_{\emptyset,0}\\
&\text{(for every } as \subseteq \{milk, sugar\} \text{ and } k \in \{0, \ldots, 399\})\ ,\\
VM1'_{\emptyset,400} &= {\sim}empty_cash_box(400) \cdot VM1'_{\emptyset,0}\ .
\end{aligned}$$

Using €0.01 as the unit of payment, we take:

$$\begin{aligned}
&\mathrm{q}({\sim}push_button(milk)) = 0\ ,\\
&\mathrm{q}({\sim}push_button(sugar)) = 0\ ,\\
&\mathrm{q}({\sim}insert_coin) = -50\ ,\\
&\mathrm{q}(deliver_coffee(as)) = 0\ ,\\
&\mathrm{q}({\sim}empty_cash_box(k)) = 50 \cdot k\ .
\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(VM1_{k_0}) = 50 \cdot k_0\ .$$

This equation indicates that $50 \cdot k_0$ is the least amount of money that is needed to account for the behaviour of $VM1_{k_0}$. This is in agreement with our intuition: no money is needed for button pushing, coin insertion and coffee delivery, $50 \cdot k_0$ units of money are needed for emptying the cash-box for the first time before the first coin insertion, $50 \cdot k_0$ units of money are also needed for emptying the cash-box for the first time after a number of coin insertions because the additional units of money spent on emptying the cash-box have been acquired by the coin insertions, and no money is needed for emptying the cash-box for a subsequent time because all units of money spent on emptying the cash-box have first been acquired by coin insertions.

We consider a user *User1* buying one cup of coffee without additives and one cup of coffee with milk and without sugar:

$$\begin{aligned}
User1 = {} & insert_coin \cdot {\sim}deliver_coffee(\emptyset) \cdot\\
& push_button(milk) \cdot insert_coin \cdot {\sim}deliver_coffee(\{milk\})\ .
\end{aligned}$$

Using €0.01 as the unit of payment, we take:

$$\begin{aligned}&\mathrm{q}(\mathit{push\_button}(\mathit{milk})) = 0\ ,\\ &\mathrm{q}(\mathit{push\_button}(\mathit{sugar})) = 0\ ,\\ &\mathrm{q}(\mathit{insert\_coin}) = 50\ ,\\ &\mathrm{q}(\sim\mathit{deliver\_coffee}(\mathit{as})) = 0\ .\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(\mathit{User1}) = 100\ .$$

This equation indicates that 100 is the least amount of money that is needed to account for the behaviour of *User1*. This is in agreement with our intuition: no money is needed for button pushing and coffee delivery, and 100 units of money are needed for the two coin insertions because no money is acquired beforehand.

We look at the process described by $\partial_{H1}(\mathit{VM1}_{k_0} \parallel \mathit{User1})$, where

$$\begin{aligned}H1 = {} & \{\mathit{push\_button}(a), \sim\mathit{push\_button}(a) \mid a \in \{\mathit{milk}, \mathit{sugar}\}\}\\ & \cup \{\mathit{insert\_coin}, \sim\mathit{insert\_coin}\}\\ & \cup \{\mathit{deliver\_coffee}(\mathit{as}), \sim\mathit{deliver\_coffee}(\mathit{as}) \mid \mathit{as} \subseteq \{\mathit{milk}, \mathit{sugar}\}\}\ .\end{aligned}$$

We can easily derive that in the case where $k_0 \leq 398$:

$$\begin{aligned}&\partial_{H1}(\mathit{VM1}_{k_0} \parallel \mathit{User1}) =\\ &\overline{\mathit{insert\_coin}} \cdot \overline{\mathit{deliver\_coffee}(\emptyset)} \cdot\\ &\overline{\mathit{push\_button}(\mathit{milk})} \cdot \overline{\mathit{insert\_coin}} \cdot \overline{\mathit{deliver\_coffee}(\{\mathit{milk}\})} \cdot\\ &\partial_{H1}(\mathit{VM1}_{k_0+2})\ .\end{aligned}$$

From this, it follows easily that

$$\mathrm{Q}(\partial_{H1}(\mathit{VM1}_{k_0} \parallel \mathit{User1})) = 50 \cdot (k_0 + 2)$$

in the case where $k_0 \leq 398$. This equation indicates that $50 \cdot (k_0 + 2)$ is the least amount of money that is needed to account for the behaviour of $\partial_{H1}(\mathit{VM1}_{k_0} \parallel \mathit{User1})$. This is in agreement with our intuition: the interactions between $\mathit{VM1}_{k_0}$ and *User1* are all CM-neutral, and after those interactions $\partial_{H1}(\mathit{VM1}_{k_0})$ behaves as $\partial_{H1}(\mathit{VM1}_{k_0+2})$.

10.2 Example 2

The second example concerns a vending machine like the one from Section 10.1. The main difference is that payments can be made by means of coins with value €0.50 and coins with value €1.00. These two kinds of coins are collected in separate cash-boxes. Coins with value €1.00 are accepted only if the cash-box for coins with value €0.50 is not empty. This vending machine, with k_0 coins in the cash-box for coins with value €0.50 and l_0 coins in the cash-box for coins with value €1.00 ($k_0, l_0 \in \{0, \ldots, 400\}$), can be defined by the guarded

recursive specification over $\mathrm{ACP}_{\mathrm{icc}}$ that consists of the following equations:

$$\begin{aligned}
VM2_{k_0,l_0} &= VM2'_{\emptyset,k_0,l_0}\\
VM2'_{as,0,l} &= {\sim}push_button(milk) \cdot VM2'_{as\cup\{milk\},0,l}\\
&\quad + {\sim}push_button(sugar) \cdot VM2'_{as\cup\{sugar\},0,l}\\
&\quad + {\sim}insert_coin(50) \cdot deliver_coffee(as) \cdot VM2'_{\emptyset,1,l}\\
&\quad + {\sim}empty_cash_box(50,0) \cdot VM2'_{\emptyset,0,l}\\
&\quad + {\sim}empty_cash_box(100,l) \cdot VM2'_{\emptyset,0,0}\\
&\quad (\text{for every } as \subseteq \{milk, sugar\} \text{ and } l \in \{0,\ldots,400\}) ,\\
VM2'_{as,k,l} &= {\sim}push_button(milk) \cdot VM2'_{as\cup\{milk\},k,l}\\
&\quad + {\sim}push_button(sugar) \cdot VM2'_{as\cup\{sugar\},k,l}\\
&\quad + {\sim}insert_coin(50) \cdot deliver_coffee(as) \cdot VM2'_{\emptyset,k+1,l}\\
&\quad + {\sim}insert_coin(100) \cdot return_coin(50)\\
&\qquad\qquad \cdot deliver_coffee(as) \cdot VM2'_{\emptyset,k-1,l+1}\\
&\quad + {\sim}empty_cash_box(50,k) \cdot VM2'_{\emptyset,0,l}\\
&\quad + {\sim}empty_cash_box(100,l) \cdot VM2'_{\emptyset,k,0}\\
&\quad (\text{for every } as \subseteq \{milk, sugar\},\ k \in \{1,\ldots,399\} \text{ and } l \in \{0,\ldots,399\}) ,\\
VM2'_{as,400,l} &= {\sim}push_button(milk) \cdot VM2'_{as\cup\{milk\},400,l}\\
&\quad + {\sim}push_button(sugar) \cdot VM2'_{as\cup\{sugar\},400,l}\\
&\quad + {\sim}insert_coin(100) \cdot return_coin(50)\\
&\qquad\qquad \cdot deliver_coffee(as) \cdot VM2'_{\emptyset,399,l+1}\\
&\quad + {\sim}empty_cash_box(50,400) \cdot VM2'_{\emptyset,0,l}\\
&\quad + {\sim}empty_cash_box(100,l) \cdot VM2'_{\emptyset,400,0}\\
&\quad (\text{for every } as \subseteq \{milk, sugar\} \text{ and } l \in \{0,\ldots,399\}) ,\\
VM2'_{\emptyset,400,400} &= {\sim}empty_cash_box(50,400) \cdot VM2'_{\emptyset,0,400}\\
&\quad + {\sim}empty_cash_box(100,400) \cdot VM2'_{\emptyset,400,0} .
\end{aligned}$$

Using €0.01 as the unit of payment, we take:

$$\begin{aligned}
&\mathrm{q}({\sim}push_button(milk)) = 0 ,\\
&\mathrm{q}({\sim}push_button(sugar)) = 0 ,\\
&\mathrm{q}({\sim}insert_coin(m)) = -m ,\\
&\mathrm{q}(return_coin(m)) = m ,\\
&\mathrm{q}(deliver_coffee(as)) = 0 ,\\
&\mathrm{q}({\sim}empty_cash_box(m,k)) = m \cdot k .
\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(VM2_{k_0,l_0}) = 50 \cdot k_0 + 100 \cdot l_0 .$$

That this equation agrees with our intuition can be seen like in the case of the equation $\mathrm{Q}(\mathit{VM1}_{k_0}) = 50 \cdot k_0$. One additional remark is in order here: no money is needed for a coin return because sufficient units of money have been acquired by the preceding coin insertion.

We consider a user *User2* buying one cup of coffee without additives and one cup of coffee with milk and without sugar using a coin with value €1.00 for the first cup of coffee and the coin with value €0.50 returned by the vending machine for the second cup of coffee:

$$\begin{aligned}&\mathit{User2} = \\ &\quad \mathit{insert\_coin}(100) \cdot {\sim}\mathit{return\_coin}(50) \cdot {\sim}\mathit{deliver\_coffee}(\emptyset) \cdot \\ &\quad \mathit{push\_button}(\mathit{milk}) \cdot \mathit{insert\_coin}(50) \cdot {\sim}\mathit{deliver\_coffee}(\{\mathit{milk}\}) \;.\end{aligned}$$

Using €0.01 as the unit of payment, we take:

$$\begin{aligned}&\mathrm{q}(\mathit{push\_button}(\mathit{milk})) = 0 \;, \\ &\mathrm{q}(\mathit{push\_button}(\mathit{sugar})) = 0 \;, \\ &\mathrm{q}(\mathit{insert\_coin}(m)) = m \;, \\ &\mathrm{q}({\sim}\mathit{return\_coin}(m)) = -m \;, \\ &\mathrm{q}({\sim}\mathit{deliver\_coffee}(\mathit{as})) = 0 \;.\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(\mathit{User2}) = 100 \;.$$

That this equation agrees with our intuition can be seen like in the case of the equation $\mathrm{Q}(\mathit{User1}) = 100$. One additional remark is in order here: 100 units of money are needed for the two coin insertions because 50 units of money is acquired by the coin return following the first coin insertion.

We look at the process described by $\partial_{H2}(\mathit{VM2}_{k_0,l_0} \parallel \mathit{User2})$, where

$$\begin{aligned}\mathit{H2} = {} &\{\mathit{push\_button}(a), {\sim}\mathit{push\_button}(a) \mid a \in \{\mathit{milk}, \mathit{sugar}\}\} \\ &\cup \{\mathit{insert\_coin}(m), {\sim}\mathit{insert\_coin}(m) \mid m \in \{50, 100\}\} \\ &\cup \{\mathit{return\_coin}(50), {\sim}\mathit{return\_coin}(50)\} \\ &\cup \{\mathit{deliver\_coffee}(\mathit{as}), {\sim}\mathit{deliver\_coffee}(\mathit{as}) \mid \mathit{as} \subseteq \{\mathit{milk}, \mathit{sugar}\}\} \;.\end{aligned}$$

We can easily derive that in the case where $k_0 \geq 1$ and $l_0 \leq 399$:

$$\begin{aligned}&\partial_{H2}(\mathit{VM2}_{k_0,l_0} \parallel \mathit{User2}) = \\ &\quad \overline{\mathit{insert\_coin}(100)} \cdot \overline{\mathit{return\_coin}(50)} \cdot \overline{\mathit{deliver\_coffee}(\emptyset)} \cdot \\ &\quad \overline{\mathit{push\_button}(\mathit{milk})} \cdot \overline{\mathit{insert\_coin}(50)} \cdot \overline{\mathit{deliver\_coffee}(\{\mathit{milk}\})} \cdot \\ &\quad \partial_{H2}(\mathit{VM2}_{k_0,l_0+1}) \;.\end{aligned}$$

From this, it follows easily that

$$\mathrm{Q}(\partial_{H2}(\mathit{VM2}_{k_0,l_0} \parallel \mathit{User2})) = 50 \cdot k_0 + 100 \cdot (l_0 + 1)$$

in the case where $k_0 \geq 1$ and $l_0 \leq 399$. That this equation agrees with our intuition can be seen like in the case of the equation $\mathrm{Q}(\partial_{H1}(\mathit{VM1}_{k_0} \parallel \mathit{User1})) = 50 \cdot (k_0 + 2)$.

We also consider a user $User2'$ buying one cup of coffee without additives and one cup of coffee with milk and without sugar using a coin with value €0.50 for the first cup of coffee and a coin with value €1.00 for the second cup of coffee:

$$\begin{aligned}&User2' = \\ &\quad insert\_coin(50) \cdot {\sim}deliver\_coffee(\emptyset) \cdot \\ &\quad push\_button(milk) \cdot \\ &\quad insert\_coin(100) \cdot {\sim}return\_coin(50) \cdot {\sim}deliver\_coffee(\{milk\})\ .\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(User2') = 150\ .$$

This shows that, although $User2'$ does not pay more for two cups of coffee than $User2$, more money is needed for the behaviour of $User2'$ than for the behaviour of $User2$. We can derive that

$$\mathrm{Q}(\partial_{H2}(VM2_{k_0,l_0} \parallel User2')) = \mathrm{Q}(\partial_{H2}(VM2_{k_0,l_0} \parallel User2))$$

in the case where $1 \leq k_0 \leq 399$ and $l_0 \leq 399$. This can be seen as follows: after the interactions between $VM2_{k_0,l_0}$ and $User2'$, as well as after the interactions between $VM2_{k_0,l_0}$ and $User2$, $\partial_{H2}(VM2_{k_0,l_0})$ behaves as $\partial_{H2}(VM2_{k_0,l_0+1})$.

10.3 Example 3

The third example concerns a vending machine like the one from Section 10.1. The main difference is that this vending machine accepts tokens instead of coins. This vending machine, with k_0 tokens in the token-box ($k_0 \in \{0,\ldots,400\}$), can be defined by the guarded recursive specification over $\mathrm{ACP}_{\mathrm{icc}}$ that consists of the following equations:

$$\begin{aligned}VM3_{k_0} &= VM3'_{\emptyset,k_0}\\ VM3'_{as,k} &= {\sim}push\_button(milk) \cdot VM3'_{as\cup\{milk\},k}\\ &\quad + {\sim}push\_button(sugar) \cdot VM3'_{as\cup\{sugar\},k}\\ &\quad + {\sim}insert\_token \cdot deliver\_coffee(as) \cdot VM3'_{\emptyset,k+1}\\ &\quad + {\sim}empty\_token\_box(k) \cdot VM3'_{\emptyset,0}\\ &\quad (\text{for every } as \subseteq \{milk, sugar\} \text{ and } k \in \{0,\ldots,399\})\ ,\\ VM3'_{\emptyset,400} &= {\sim}empty\_token\_box(400) \cdot VM3'_{\emptyset,0}\ .\end{aligned}$$

We might take the view that tokens are a form of money. In that case, there would be no essential difference between the example from Section 10.1 and this one. Here, we take the view that tokens are not a form of money. This

means that, using €0.01 as the unit of payment, we take:

$$\begin{aligned}
&\mathrm{q}(\sim push_button(milk)) = 0 \;,\\
&\mathrm{q}(\sim push_button(sugar)) = 0 \;,\\
&\mathrm{q}(\sim insert_token) = 0 \;,\\
&\mathrm{q}(deliver_coffee(as)) = 0 \;,\\
&\mathrm{q}(\sim empty_token_box(k)) = 0 \;.
\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(VM3_{k_0}) = 0 \;.$$

This equation indicates that 0 is the least amount of money that is needed to account for the behaviour of $VM3_{k_0}$. This is in agreement with our intuition: like in the case of $VM1_{k_0}$, no money is needed for button pushing, token insertion and coffee delivery, and moreover no money is needed for emptying the token-box because tokens are not considered as money.

The tokens are obtained from a token-issuing machine. This is a machine by which tokens can be bought. The price of a token is €0.50. Payments can be made by means of coins with value €0.50 only. The coins are collected in a cash-box with a capacity of 400 coins that can be emptied at any time. The token-issuing machine, with l_0 coins in the cash-box ($l_0 \in \{0, \ldots, 400\}$), can be defined by the guarded recursive specification over $\mathrm{ACP}_{\mathrm{icc}}$ that consists of the following equations:

$$\begin{aligned}
TM_{l_0} &= TM'_{l_0}\\
TM_l &= \sim insert_coin \cdot deliver_token \cdot TM_{l+1}\\
&\quad + \sim empty_cash_box(l) \cdot TM_0\\
&\text{(for every } l \in \{0, \ldots, 399\}) \;,\\
TM_{400} &= \sim empty_cash_box(400) \cdot TM_0 \;.
\end{aligned}$$

Using €0.01 as the unit of payment, we take:

$$\begin{aligned}
&\mathrm{q}(\sim insert_coin) = -50 \;,\\
&\mathrm{q}(deliver_token) = 0 \;,\\
&\mathrm{q}(\sim empty_cash_box(l)) = 50 \cdot l \;.
\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(TM_{l_0}) = 50 \cdot l_0 \;.$$

That this equation agrees with our intuition can be seen like in the case of the equation $\mathrm{Q}(VM1_{k_0}) = 50 \cdot k_0$ because TM_{l_0} is essentially the same as $VM1_{l_0}$, but tokens are sold instead of cups of coffee.

We consider a user *User3* buying one cup of coffee without additives and

one cup of coffee with milk and without sugar:

$$\begin{aligned}&\mathit{User3} =\\ &\quad \mathit{insert\_coin} \cdot {\sim}\mathit{deliver\_token} \cdot \mathit{insert\_coin} \cdot {\sim}\mathit{deliver\_token} \cdot\\ &\quad \mathit{insert\_token} \cdot {\sim}\mathit{deliver\_coffee}(\emptyset) \cdot\\ &\quad \mathit{push\_button}(\mathit{milk}) \cdot \mathit{insert\_token} \cdot {\sim}\mathit{deliver\_coffee}(\{\mathit{milk}\})\ .\end{aligned}$$

Using €0.01 as the unit of payment, we take:

$$\begin{aligned}&\mathrm{q}(\mathit{insert\_coin}) = 50\ ,\\ &\mathrm{q}({\sim}\mathit{deliver\_token}) = 0\ ,\\ &\mathrm{q}(\mathit{push\_button}(\mathit{milk})) = 0\ ,\\ &\mathrm{q}(\mathit{push\_button}(\mathit{sugar})) = 0\ ,\\ &\mathrm{q}(\mathit{insert\_token}) = 0\ ,\\ &\mathrm{q}({\sim}\mathit{deliver\_coffee}(\mathit{as})) = 0\ .\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(\mathit{User3}) = 100\ .$$

That this equation agrees with our intuition can be seen like in the case of the equation $\mathrm{Q}(\mathit{User1}) = 100$ because *User3* is essentially the same as *User1*, but tokens are bought instead of cups of coffee and then exchanged for cups of coffee.

We look at the process described by $\partial_{H3}(\mathit{VM3}_{k_0} \parallel \mathit{TM}_{l_0} \parallel \mathit{User3})$, where

$$\begin{aligned}\mathit{H3} = {}&\{\mathit{insert\_coin}, {\sim}\mathit{insert\_coin}\}\\ &\cup \{\mathit{deliver\_token}, {\sim}\mathit{deliver\_token}\}\\ &\cup \{\mathit{push\_button}(a), {\sim}\mathit{push\_button}(a) \mid a \in \{\mathit{milk}, \mathit{sugar}\}\}\\ &\cup \{\mathit{insert\_token}, {\sim}\mathit{insert\_token}\}\\ &\cup \{\mathit{deliver\_coffee}(\mathit{as}), {\sim}\mathit{deliver\_coffee}(\mathit{as}) \mid \mathit{as} \subseteq \{\mathit{milk}, \mathit{sugar}\}\}\ .\end{aligned}$$

We can easily derive that in the case where $k_0 \leq 398$ and $l_0 \leq 398$:

$$\begin{aligned}&\partial_{H3}(\mathit{VM3}_{k_0} \parallel \mathit{TM}_{l_0} \parallel \mathit{User3}) =\\ &\quad \overline{\mathit{insert\_coin}} \cdot \overline{\mathit{deliver\_token}} \cdot \overline{\mathit{insert\_coin}} \cdot \overline{\mathit{deliver\_token}} \cdot\\ &\quad \overline{\mathit{insert\_token}} \cdot \overline{\mathit{deliver\_coffee}(\emptyset)} \cdot\\ &\quad \overline{\mathit{push\_button}(\mathit{milk})} \cdot \overline{\mathit{insert\_token}} \cdot \overline{\mathit{deliver\_coffee}(\{\mathit{milk}\})} \cdot\\ &\quad \partial_{H3}(\mathit{VM3}_{k_0+2} \parallel \mathit{TM}_{l_0+2})\end{aligned}$$

From this, it follows easily that

$$\mathrm{Q}(\partial_{H3}(\mathit{VM3}_{k_0} \parallel \mathit{TM}_{l_0} \parallel \mathit{User3})) = 50 \cdot (l_0 + 2)$$

in the case where $k_0 \leq 398$ and $l_0 \leq 398$. That this equation agrees with our intuition can be seen like in the case of the equation $\mathrm{Q}(\partial_{H1}(\mathit{VM1}_{k_0} \parallel \mathit{User1})) = 50 \cdot (k_0 + 2)$. One additional remark is in order here: there are no interactions between $\mathit{VM3}_{k_0}$ and TM_{l_0}.

10.4 Example 4

Chipcards with an electronic wallet appear to provide an electronic pendant of tokens. The fourth example concerns a vending machine like the one from Section 10.3. The main difference is that it accepts this electronic pendant of tokens instead of tokens.

We take the view that, unlike tokens, the electronic pendant of tokens provided by a chipcard with an electronic wallet constitutes a form of money. The reason for this is that it is considered and used as money by a large group of people. The electronic payments are collected in an electronic cash-box. The value of the electronic money in the electronic cash-box is at most €1000. Assuming that the value of the electronic money in the electronic wallet on a chipcard is at most €100, the vending machine, with the amount m_0 of electronic money in the electronic cash-box ($m_0 \in \{0, \ldots, 100000\}$), can be defined by the guarded recursive specification over $\mathrm{ACP}_{\mathrm{icc}}$ that consists of the following equations:

$$\begin{aligned}
VM4_{m_0} &= VM4'_{\emptyset,m_0} \;,\\
VM4'_{as,m} &= {\sim}push_button(milk) \cdot VM4'_{as\cup\{milk\},m}\\
&\quad + {\sim}push_button(sugar) \cdot VM4'_{as\cup\{sugar\},m}\\
&\quad + \sum_{\substack{m'\in M\\ m'\geq 50}} {\sim}insert_chipcard(m') \cdot return_chipcard(m'-50)\\
&\qquad\qquad \cdot deliver_coffee(as) \cdot VM4'_{\emptyset,m+50}\\
&\quad + {\sim}empty_ecash_box(m) \cdot VM4'_{\emptyset,0}\\
&\text{(for every } as \subseteq \{milk, sugar\} \text{ and } m \in \{0, \ldots, 99950\}) \;,\\
VM4'_{\emptyset,100000} &= {\sim}empty_ecash_box(100000) \cdot VM4'_{\emptyset,0} \;,
\end{aligned}$$

In this specification and forthcoming specifications, we write M for $\{m \in \mathbf{M} \mid 0 \leq m \leq 10000\}$. Using €0.01 as the unit of payment, M is the set of possible values of the electronic money in the electronic wallet on a chipcard. Using €0.01 as the unit of payment, we take:

$$\begin{aligned}
&\mathrm{q}({\sim}push_button(milk)) = 0 \;,\\
&\mathrm{q}({\sim}push_button(sugar)) = 0 \;,\\
&\mathrm{q}({\sim}insert_chipcard(m)) = -m \;,\\
&\mathrm{q}(return_chipcard(m)) = m \;,\\
&\mathrm{q}(deliver_coffee(as)) = 0 \;,\\
&\mathrm{q}({\sim}empty_ecash_box(m)) = m \;.
\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(VM4_{m_0}) = m_0 \;.$$

That this equation agrees with our intuition can be seen like in the case of the equation $\mathrm{Q}(VM1_{k_0}) = 50 \cdot k_0$. One additional remark is in order here: no money is needed for returning the chipcard because sufficient units of money

have been acquired by the preceding chipcard insertion.

Electronic money is loaded into a chipcard by means of an electronic cash dispenser. The value of the electronic money that can be loaded at a time is €10. Moreover, no electronic money can be loaded if the balance of the bank account associated with the chipcard is less than €10 or the value of the electronic money in the electronic wallet would exceed €100 after loading.

Seeing that the behaviour of an electronic cash dispenser depends upon the balance of the bank accounts for which chipcards with an electronic wallet have been issued, the balance of all those accounts together make up a parameter of the electronic cash dispenser. We formalize this parameter using account numbers. We assume that a finite set N of account numbers has been given. N is considered to be the set of all account numbers of bank accounts for which chipcards with an electronic wallet have been issued. The parameter is formalized by a function $ba : N \to \mathbf{M}$. This function is considered to map each account number to the balance of the bank account with that account number. We write BA for the set of all functions $ba : N \to \mathbf{M}$.

The electronic cash dispenser, parametrized by $ba_0 \in BA$, can be defined by the guarded recursive specification over $\mathrm{ACP}_{\mathrm{icc}}$ that consists of the following equations: [6]

$$
\begin{aligned}
ECD_{ba_0} &= ECD'_{ba_0} \ , \\
ECD'_{ba} &= \sum_{\substack{n \in N \\ ba(n) \geq 1000}} \sum_{\substack{m \in M \\ m+1000 \in M}} {\sim}insert\_chipcard(n,m) \\
&\qquad \cdot return\_chipcard(m+1000) \cdot ECD'_{ba \oplus [n \mapsto ba(n)-1000]} \\
&\quad + \sum_{\substack{n \in N \\ ba(n) < 1000}} \sum_{\substack{m \in M \\ m+1000 \in M}} {\sim}insert\_chipcard(n,m) \\
&\qquad \cdot return\_chipcard(m) \cdot ECD'_{ba} \\
&\quad + \sum_{n \in N} \sum_{\substack{m \in M \\ m+1000 \notin M}} {\sim}insert\_chipcard(n,m) \\
&\qquad \cdot return\_chipcard(m) \cdot ECD'_{ba} \ .
\end{aligned}
$$

Using €0.01 as the unit of payment, we take:

$$
\begin{aligned}
\mathrm{q}({\sim}insert\_chipcard(n,m)) &= -m \ , \\
\mathrm{q}(return\_chipcard(m)) &= m \ .
\end{aligned}
$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(ECD_{ba_0}) = 1000 \cdot \textstyle\sum_{n \in N} \max\{k \mid 1000 \cdot k \leq ba_0(n)\} \ .$$

This equation indicates that $1000 \cdot \sum_{n \in N} \max\{k \mid 1000 \cdot k \leq ba_0(n)\}$ is the least amount of money that is needed to account for the behaviour of ECD_{ba_0}.

[6] We use the following notation for functions: $[\,]$ for the empty function; $[d \mapsto r]$ for the function f with $\mathrm{dom}(f) = \{d\}$ such that $f(d) = r$; $f \oplus g$ for the function h with $\mathrm{dom}(h) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$ such that for all $d \in \mathrm{dom}(h)$, $h(d) = f(d)$ if $d \notin \mathrm{dom}(g)$ and $h(d) = g(d)$ otherwise.

This is in agreement with our intuition: money is needed only for returning a chipcard with more electronic money in the electronic wallet than on the preceding insertion of the chipcard, 1000 units of money are needed for that, and it can take place for every chipcard with an electronic wallet issued as many times as allowed by the balance of the associated bank account.

We consider a user $\mathit{User4}_{n_0}$ ($n_0 \in N$), with an empty electronic wallet on his or her chipcard, buying one cup of coffee without additives and one cup of coffee with milk and without sugar:

$$\begin{aligned}
&\mathit{User4}_{n_0} = \\
&\quad \mathit{insert_chipcard}(n_0, 0) \cdot {\sim}\mathit{return_chipcard}(1000) \cdot \\
&\quad \mathit{insert_chipcard}(1000) \cdot {\sim}\mathit{return_chipcard}(950) \cdot {\sim}\mathit{deliver_coffee}(\emptyset) \cdot \\
&\quad \mathit{push_button}(\mathit{milk}) \cdot \\
&\quad \mathit{insert_chipcard}(950) \cdot {\sim}\mathit{return_chipcard}(900) \cdot {\sim}\mathit{deliver_coffee}(\{\mathit{milk}\}) \;.
\end{aligned}$$

Using €0.01 as the unit of payment, we take:

$$\begin{aligned}
&\mathrm{q}(\mathit{insert_chipcard}(n, m)) = m \;, \\
&\mathrm{q}({\sim}\mathit{return_chipcard}(m)) = -m \;, \\
&\mathrm{q}(\mathit{push_button}(\mathit{milk})) = 0 \;, \\
&\mathrm{q}(\mathit{push_button}(\mathit{sugar})) = 0 \;, \\
&\mathrm{q}(\mathit{insert_chipcard}(m)) = m \;, \\
&\mathrm{q}({\sim}\mathit{deliver_coffee}(as)) = 0 \;.
\end{aligned}$$

We can prove by means of axioms ICC1–ICC5 that

$$\mathrm{Q}(\mathit{User4}_{n_0}) = 0 \;.$$

This equation indicates that 0 is the least amount of money that is needed to account for the behaviour of $\mathit{User4}_{n_0}$. This is in agreement with our intuition: no money is needed for the first chipcard insertion because it concerns an empty electronic wallet, and no money is needed for the second and third chipcard insertion because enough money has been acquired by the preceding chipcard return. Notice that $\mathit{User4}_{n_0}$ would have an implicit computational capital greater than zero if he or she would start with a non-empty electronic wallet.

We look at the process described by $\partial_{H4}(\mathit{VM4}_{m_0} \parallel \mathit{ECD}_{ba_0} \parallel \mathit{User4}_{n_0})$, where

$$\begin{aligned}
H4 = {} & \{\mathit{insert_chipcard}(n, m), {\sim}\mathit{insert_chipcard}(n, m) \mid n \in N \wedge m \in M\} \\
& \cup \{\mathit{return_chipcard}(m), {\sim}\mathit{return_chipcard}(m) \mid m \in M\} \\
& \cup \{\mathit{push_button}(a), {\sim}\mathit{push_button}(a) \mid a \in \{\mathit{milk}, \mathit{sugar}\}\} \\
& \cup \{\mathit{insert_chipcard}(m), {\sim}\mathit{insert_chipcard}(m) \mid m \in M\} \\
& \cup \{\mathit{deliver_coffee}(as), {\sim}\mathit{deliver_coffee}(as) \mid as \subseteq \{\mathit{milk}, \mathit{sugar}\}\} \;.
\end{aligned}$$

We can easily derive that in the case where $ba_0(n_0) \geq 1000$:

$$\begin{aligned}
&\partial_{H4}(VM4_{m_0} \parallel ECD_{ba_0} \parallel User4_{n_0}) = \\
&\quad \overline{insert_chipcard(n_0, 0)} \cdot \overline{return_chipcard(1000)} \cdot \\
&\quad \overline{insert_chipcard(1000)} \cdot \overline{return_chipcard(950)} \cdot \overline{deliver_coffee(\emptyset)} \cdot \\
&\quad \overline{push_button(milk)} \cdot \\
&\quad \overline{insert_chipcard(950)} \cdot \overline{return_chipcard(900)} \cdot \overline{deliver_coffee(\{milk\})} \cdot \\
&\quad \partial_{H4}(VM4_{m_0+100} \parallel ECD_{ba_0 \oplus [n_0 \mapsto ba_0(n_0)-1000]}) \cdot
\end{aligned}$$

From this, it follows easily that

$$\begin{aligned}
&\mathrm{Q}(\partial_{H4}(VM4_{m_0} \parallel ECD_{ba_0} \parallel User4_{n_0})) = \\
&\quad m_0 + 100 + 1000 \cdot \textstyle\sum_{n \in N} \max\{k \mid 1000 \cdot k \leq ba'_0(n)\} \,,
\end{aligned}$$

where $ba'_0 = ba_0 \oplus [n_0 \mapsto ba_0(n_0) - 1000]$, in the case where $ba_0(n_0) \geq 1000$. That this equation agrees with our intuition can be seen like in the case of the equation $\mathrm{Q}(\partial_{H1}(VM1_{k_0} \parallel User1)) = 50 \cdot (k_0 + 2)$. One additional remark is in order here: there are no interactions between $VM4_{m_0}$ and ECD_{ba_0}.

10.5 Closing Remark on the Examples

Vending machines that are more realistic than the ones treated in the examples given above can easily be devised, e.g.:

- vending machines where several products can be bought, and different products may have different prices;
- vending machines where payments can be made by means of more than two kinds of coins;
- vending machines where the products do not have to be paid one at a time;
- vending machines where payments can be made using coins as well as electronic forms of money;
- vending machines with a bounded stock of the products that can be bought;
- vending machines with a proper user interface.

We believe that additional examples concerning more realistic vending machines do not add to a better understanding of the notion of implicit computational capital of a process. A realistic vending machines might be the subject of a first case-study to assess the degree of usefulness of $\mathrm{ACP}_{\mathrm{icc}}$ in practical applications, but such a case study is considered outside the scope of this paper.

11 Conclusions

In this paper, we build on earlier work on ACP. The algebraic theory ACP was first presented in [3], abstraction from internal actions was added in [4], and RDP, RSP and AIP were first formulated in [5]. Transition systems as

defined in this paper are basically the process graphs from early work on ACP, which are most extensively described in [1]. The notion of transition system induced by a model of ACP was recently introduced in [6].

To the best of our knowledge, there is no related work. Many options for future work remain. We mention:

- investigations into plausible mechanisms to model preferences in spending and acquiring computational money;
- investigations into plausible mechanisms to reveal the implicit computational capital of a process to other processes;
- investigations into plausible mechanisms to transform some of the actions of infinite processes that can perform CM-neutral actions only into non-CM-neutral actions in a way that prevents the resulting processes from becoming money sources;
- investigations into decidability and computability issues concerning Q.

The work presented in this paper was carried out in the framework of a project investigating IT sourcing. In that project, we look at IT sourcing from the perspective of formal methods. IT sourcing is mainly actuated by financial issues. We believe that the idea of processes with an implicit computational capital may be helpful to understand those financial issues.

References

[1] Baeten, J. C. M., J. A. Bergstra and J. W. Klop, *On the consistency of Koomen's fair abstraction rule*, Theoretical Computer Science **51** (1987), pp. 129–176.

[2] Baeten, J. C. M. and W. P. Weijland, "Process Algebra," Cambridge Tracts in Theoretical Computer Science **18**, Cambridge University Press, Cambridge, 1990.

[3] Bergstra, J. A. and J. W. Klop, *Process algebra for synchronous communication*, Information and Control **60** (1984), pp. 109–137.

[4] Bergstra, J. A. and J. W. Klop, *Algebra of communicating processes with abstraction*, Theoretical Computer Science **37** (1985), pp. 77–121.

[5] Bergstra, J. A. and J. W. Klop, *Process algebra: Specification and verification in bisimulation semantics*, in: M. Hazewinkel, J. K. Lenstra and L. G. L. T. Meertens, editors, *Proceedings Mathematics and Computer Science II*, CWI Monograph **4** (1986), pp. 61–94.

[6] Bergstra, J. A. and C. A. Middelburg, *Model theory for process algebra*, in: A. Middeldorp, V. van Oostrom, F. van Raamsdonk and R. C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity*, Lecture Notes in Computer Science **3838** (2005), pp. 445–495.

[7] Bergstra, J. A. and J. V. Tucker, *Elementary algebraic specification of the rational complex numbers*, in: K. Futatsugi et al., editors, *Goguen Festschrift*, Lecture Notes in Computer Science **4060** (2006), pp. 459–475.

[8] Hodges, W. A., "Model Theory," Encyclopedia of Mathematics and Its Applications **42**, Cambridge University Press, Cambridge, 1993.

[9] Milner, R., "A Calculus of Communicating Systems," Lecture Notes in Computer Science **92**, Springer-Verlag, Berlin, 1980.

[10] Milner, R., "Communication and Concurrency," Prentice-Hall, Englewood Cliffs, 1989.

[11] Sannella, D. and A. Tarlecki, *Algebraic preliminaries*, in: E. Astesiano, H.-J. Kreowski and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, Springer-Verlag, Berlin, 1999, pp. 13–30.

[12] van Glabbeek, R. J. and W. P. Weijland, *Branching time and abstraction in bisimulation semantics (extended abstract)*, in: G. X. Ritter, editor, *Information Processing 89* (1989), pp. 613–618, see [13] for the full version.

[13] van Glabbeek, R. J. and W. P. Weijland, *Branching time and abstraction in bisimulation semantics*, Journal of the ACM **43** (1996), pp. 555–600.

[14] Wirsing, M., *Algebraic specification*, in: J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, **B**, Elsevier, Amsterdam, 1990, pp. 675–788.

Electronic Reports Series of the Programming Research Group

Within this series the following reports appeared.

[PRG0609] B. Diertens, *Software (Re-)Engineering with PSF II: from architecture to implementation*, Programming Research Group - University of Amsterdam, 2006.

[PRG0608] A. Ponse and M.B. van der Zwaag, *Risk Assessment for One-Counter Threads*, Programming Research Group - University of Amsterdam, 2006.

[PRG0607] J.A. Bergstra and C.A. Middelburg, *Synchronous Cooperation for Explicit Multi-Threading*, Programming Research Group - University of Amsterdam, 2006.

[PRG0606] J.A. Bergstra and M. Burgess, *Local and Global Trust Based on the Concept of Promises*, Programming Research Group - University of Amsterdam, 2006.

[PRG0605] J.A. Bergstra and J.V. Tucker, *Division Safe Calculation in Totalised Fields*, Programming Research Group - University of Amsterdam, 2006.

[PRG0604] J.A. Bergstra and A. Ponse, *Projection Semantics for Rigid Loops*, Programming Research Group - University of Amsterdam, 2006.

[PRG0603] J.A. Bergstra and I. Bethke, *Predictable and Reliable Program Code: Virtual Machine-based Projection Semantics (submitted for inclusion in the Handbook of Network and Systems Administration)*, Programming Research Group - University of Amsterdam, 2006.

[PRG0602] J.A. Bergstra and A. Ponse, *Program Algebra with Repeat Instruction*, Programming Research Group - University of Amsterdam, 2006.

[PRG0601] J.A. Bergstra and A. Ponse, *Interface Groups for Analytic Execution Architectures*, Programming Research Group - University of Amsterdam, 2006.

[PRG0505] B. Diertens, *Software (Re-)Engineering with PSF*, Programming Research Group - University of Amsterdam, 2005.

[PRG0504] P.H. Rodenburg, *Piecewise Initial Algebra Semantics*, Programming Research Group - University of Amsterdam, 2005.

[PRG0503] T.D. Vu, *Metric Denotational Semantics for BPPA*, Programming Research Group - University of Amsterdam, 2005.

[PRG0502] J.A. Bergstra, I. Bethke, and A. Ponse, *Decision Problems for Pushdown Threads*, Programming Research Group - University of Amsterdam, 2005.

[PRG0501] J.A. Bergstra and A. Ponse, *A Bypass of Cohen's Impossibility Result*, Programming Research Group - University of Amsterdam, 2005.

[PRG0405] J.A. Bergstra and I. Bethke, *An Upper Bound for the Equational Specification of Finite State Services*, Programming Research Group - University of Amsterdam, 2004.

[PRG0404] J.A. Bergstra and C.A. Middelburg, *Thread Algebra for Strategic Interleaving*, Programming Research Group - University of Amsterdam, 2004.

[PRG0403] B. Diertens, *A Compiler-projection from PGLEc.MSPio to Parrot*, Programming Research Group - University of Amsterdam, 2004.

[PRG0402] J.A. Bergstra and I. Bethke, *Linear Projective Program Syntax*, Programming Research Group - University of Amsterdam, 2004.

[PRG0401] B. Diertens, *Molecular Scripting Primitives*, Programming Research Group - University of Amsterdam, 2004.

[PRG0302] B. Diertens, *A Toolset for PGA*, Programming Research Group - University of Amsterdam, 2003.

[PRG0301] J.A. Bergstra and P. Walters, *Projection Semantics for Multi-File Programs*, Programming Research Group - University of Amsterdam, 2003.

[PRG0201] I. Bethke and P. Walters, *Molecule-oriented Java Programs for Cyclic Sequences*, Programming Research Group - University of Amsterdam, 2002.

The above reports and more are available through the website: www.science.uva.nl/research/prog/

Electronic Report Series

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
the Netherlands

www.science.uva.nl/research/prog/