Imperative process algebra with abstraction

Middelburg, C.A.

DOI
10.48550/arXiv.2103.07863

Publication date
2021

Document Version
Submitted manuscript

Imperative Process Algebra with Abstraction

C.A. Middelburg

Informatics Institute, Faculty of Science, University of Amsterdam
Science Park 904, 1098 XH Amsterdam, the Netherlands
C.A.Middelburg@uva.nl

Abstract. This paper introduces an imperative process algebra based on ACP (Algebra of Communicating Processes). Like other imperative process algebras, this process algebra deals with processes of the kind that arises from the execution of imperative programs. It distinguishes itself from already existing imperative process algebras among other things by supporting abstraction from actions that are considered not to be visible. The support of abstraction opens interesting application possibilities of the process algebra. This paper goes briefly into the possibility of information-flow security analysis of the kind that is concerned with the leakage of confidential data. For the presented axiomatization, soundness and semi-completeness results with respect to a notion of branching bisimulation equivalence are established.

Keywords: imperative process algebra, abstraction, branching bisimulation, information-flow security, data non-interference with interactions.

1998 ACM Computing Classification: D.2.4, D.4.6, F.1.1, F.1.2, F.3.1.

1 Introduction

Generally speaking, process algebras focus on the main role of a reactive system, namely maintaining a certain ongoing interaction with its environment. Reactive systems contrast with transformational systems. A transformational system is a system whose main role is to produce, without interruption by its environment, output data from input data.[1] In general, early computer-based systems were transformational systems. Nowadays, many systems are composed of both reactive components and transformational components. A process carried out by such a system is a process in which data are involved. Usually, the data change in the course of the process, the process proceeds at certain stages in a way that depends on the changing data, and the interaction of the process with other processes consists of communication of data.

This paper introduces an extension of ACP [6] with features that are relevant to processes of the kind referred to above. The extension concerned is called $\mathrm{ACP}_{\epsilon}^{\tau}$-I. Its additional features include assignment actions to change the data in the course of a process, guarded commands to proceed at certain stages of a

[1] The terms reactive system and transformational system are used here with the meaning given in [27].

process in a way that depends on the changing data, and data parameterized actions to communicate data between processes. The processes of the kind that $\mathrm{ACP}_\epsilon^\tau$-I is concerned with are reminiscent of the processes that arise from the execution of imperative programs. In [34], the term imperative process algebra was coined for process algebras like $\mathrm{ACP}_\epsilon^\tau$-I. Other imperative process algebras are VPLA [28], IPAL [34], CSP_σ [16], AWN [19], and the unnamed process algebra introduced in [13].

$\mathrm{ACP}_\epsilon^\tau$-I distinguishes itself from those imperative process algebras by being the only one with the following three properties: (a) it supports abstraction from actions that are considered not to be visible; (b) a verification of the equivalence of two processes in its semantics is automatically valid in any semantics that is fully abstract with respect to some notion of observable behaviour (cf. [24]); (c) it offers the possibility of equational verification of process equivalence due to an equational axiomatization of the equivalence concerned. Properties (a) and (b) are essential and desirable, respectively, for an imperative process algebra suitable for analysis of information-flow security. CSP_σ is the only one of the above-mentioned imperative process algebras that has property (a) and none of them have property (b). $\mathrm{ACP}_\epsilon^\tau$-I is probably unique in being the only imperative process algebra with properties (a), (b) and (c).

A great part of the work done on information-flow security is concerned with secure information flow in programs, where information flow in a program is considered secure if information derivable from the confidential data contained in its high-security variables cannot be inferred from the non-confidential data contained in its low-security variables (see e.g. [42,41,12,35,10]). A notable exception is the work done in a process-algebra setting, where the focus has shifted from programs to processes of the kind to which programs in execution belong and the information flow in a process is usually considered secure if information derivable from confidential actions cannot be inferred from non-confidential actions (see e.g. [20,37,11,32]).

Recently, work done on information-flow security in a process-algebra setting occasionally deals with the data-oriented notion of secure information flow, but on such occasions program variables are always mimicked by processes (see e.g. [21,30]). A suitable imperative process algebra would obviate the need to mimic program variables. This state of affairs motivated the development of $\mathrm{ACP}_\epsilon^\tau$-I. This paper also shows how $\mathrm{ACP}_\epsilon^\tau$-I can be used for information-flow security analysis of the kind that is concerned with the leakage of confidential data.

$\mathrm{ACP}_\epsilon^\tau$-I is closely related to ACP_ϵ^*-D [7]. The main differences between them can be summarized as follows: (a) only the former supports abstraction from actions that are considered not to be visible and (b) only the latter has an iteration operator. This paper introduces, in addition to $\mathrm{ACP}_\epsilon^\tau$-I, guarded linear recursion in the setting of $\mathrm{ACP}_\epsilon^\tau$-I. The set of processes that can be defined by means of the operators of $\mathrm{ACP}_\epsilon^\tau$-I extended with the iteration operator is a proper subset of the set of processes that can be defined by means of guarded linear recursion in the setting of $\mathrm{ACP}_\epsilon^\tau$-I. Therefore, (a) should be considered the important difference.

This paper is organized as follows. First, a survey of the algebraic theory $\mathrm{ACP}_\epsilon^\tau$, which is the extension of ACP with the empty process constant ϵ and the silent step constant τ, is given (Section 2). Next, the algebraic theory $\mathrm{ACP}_\epsilon^\tau$-I is introduced as an extension of $\mathrm{ACP}_\epsilon^\tau$ (Section 3) and guarded linear recursion in the setting of $\mathrm{ACP}_\epsilon^\tau$-I is treated (Section 4). After that, a structural operational semantics of $\mathrm{ACP}_\epsilon^\tau$-I is presented and a notion of branching bisimulation equivalence based on this semantics is defined (Section 5). Following this, the reasons for two relatively uncommon choices made in the preceding sections are clarified (Section 6). Then, results concerning the soundness and (semi-)completeness of the given axiomatization with respect to branching bisimulation equivalence are established (Section 7). Thereafter, it is explained how $\mathrm{ACP}_\epsilon^\tau$-I can be used for information-flow security analysis of the kind that is concerned with the leakage of confidential data (Section 8). Finally, some concluding remarks are made (Section 9). There is also an appendix in which, for comparison, an alternative structural operational semantics of $\mathrm{ACP}_\epsilon^\tau$-I is presented and a notion of branching bisimulation equivalence based on this alternative structural operational semantics is defined.

Section 2, Section 3, and the appendix mainly extend the material in Section 2, Section 3, and Section 4, respectively, of [7]. Portions of that material have been copied near verbatim or slightly modified.

2 ACP with Empty Process and Silent Step

In this section, $\mathrm{ACP}_\epsilon^\tau$ is presented. $\mathrm{ACP}_\epsilon^\tau$ is ACP [6] extended with the empty process constant ϵ and the silent step constant τ as in [3, Section 5.3]. In $\mathrm{ACP}_\epsilon^\tau$, it is assumed that a fixed but arbitrary finite set A of *basic actions*, with $\tau, \delta, \epsilon \notin \mathsf{A}$, and a fixed but arbitrary commutative and associative *communication* function $\gamma : (\mathsf{A} \cup \{\tau, \delta\}) \times (\mathsf{A} \cup \{\tau, \delta\}) \to (\mathsf{A} \cup \{\tau, \delta\})$, such that $\gamma(\tau, a) = \delta$ and $\gamma(\delta, a) = \delta$ for all $a \in \mathsf{A} \cup \{\tau, \delta\}$, have been given. Basic actions are taken as atomic processes. The function γ is regarded to give the result of synchronously performing any two basic actions for which this is possible, and to be δ otherwise. Henceforth, we write A_τ for $\mathsf{A} \cup \{\tau\}$ and $\mathsf{A}_{\tau\delta}$ for $\mathsf{A} \cup \{\tau, \delta\}$.

The algebraic theory $\mathrm{ACP}_\epsilon^\tau$ has one sort: the sort $\mathbf{P}$ of *processes*. This sort is made explicit to anticipate the need for many-sortedness later on. The algebraic theory $\mathrm{ACP}_\epsilon^\tau$ has the following constants and operators to build terms of sort $\mathbf{P}$:

- for each $a \in \mathsf{A}$, the *basic action* constant $a : \to \mathbf{P}$;
- the *silent step* constant $\tau : \to \mathbf{P}$;
- the *inaction* constant $\delta : \to \mathbf{P}$;
- the *empty process* constant $\epsilon : \to \mathbf{P}$;
- 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 $\mathbin{⌊\!⌊} : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;

– the binary *communication merge* operator $\mid : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
– for each $H \subseteq \mathsf{A}$, the unary *encapsulation* operator $\partial_H : \mathbf{P} \to \mathbf{P}$;
– for each $I \subseteq \mathsf{A}$, the unary *abstraction* operator $\tau_I : \mathbf{P} \to \mathbf{P}$.

It is assumed that there is a countably infinite set $\mathcal{X}$ of variables of sort $\mathbf{P}$, which contains x, y and z. Terms are built as usual. Infix notation is used for the binary operators. The following precedence conventions are used to reduce the need for parentheses: the operator $\cdot$ binds stronger than all other binary operators and the operator $+$ binds weaker than all other binary operators.

The constants of $\mathrm{ACP}_\epsilon^\tau$ can be explained as follows ($a \in \mathsf{A}$):

– a denotes the process that performs the observable action a and after that terminates successfully;
– τ denotes the process that performs the unobservable action τ and after that terminates successfully;
– ϵ denotes the process that terminates successfully without performing any action;
– δ denotes the process that cannot do anything, it cannot even terminate successfully.

Let t and t' be closed $\mathrm{ACP}_\epsilon^\tau$ terms denoting processes p and p', and let $H, I \subseteq \mathsf{A}$. Then the operators of $\mathrm{ACP}_\epsilon^\tau$ can be explained as follows:

– $t + t'$ denotes the process that behaves either as p or as p', where the choice between the two is resolved at the instant that one of them performs its first action, and not before;
– $t \cdot t'$ denotes the process that first behaves as p and following successful termination of p behaves as p';
– $t \parallel t'$ denotes the process that behaves as p and p' in parallel, by which is meant that, each time an action is performed, either a next action of p is performed or a next action of p' is performed or a next action of p and a next action of p' are performed synchronously;
– $t \mathbin{\|\!\_} t'$ denotes the same process as $t \parallel t'$, except that it starts with performing an action of p;
– $t \mid t'$ denotes the same process as $t \parallel t'$, except that it starts with performing an action of p and an action of p' synchronously;
– $\partial_H(t)$ denotes the process that behaves the same as p, except that actions from H are blocked from being performed;
– $\tau_I(t)$ denotes the process that behaves the same as p, except that actions from I are turned into the unobservable action τ.

The axioms of $\mathrm{ACP}_\epsilon^\tau$ are presented in Table 1. In these equations, a, b, and α stand for arbitrary constants of $\mathrm{ACP}_\epsilon^\tau$ other than ϵ; and H and I stand for arbitrary subsets of A. So, CM3, CM7, D0–D4, T0–T4, and BE are actually axiom schemas. In this paper, axiom schemas will usually be referred to as axioms.

Table 1. Axioms of $\mathrm{ACP}_\epsilon^\tau$

$x + y = y + x$		A1
$(x + y) + z = x + (y + z)$		A2
$x + x = x$		A3
$(x + y) \cdot z = x \cdot z + y \cdot z$		A4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$		A5
$x + \delta = x$		A6
$\delta \cdot x = \delta$		A7
$x \cdot \epsilon = x$		A8
$\epsilon \cdot x = x$		A9
$x \parallel y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y + \partial_{\mathsf{A}}(x) \cdot \partial_{\mathsf{A}}(y)$		CM1E
$\epsilon \mathbin{\lfloor\!\lfloor} x = \delta$		CM2E
$\alpha \cdot x \mathbin{\lfloor\!\lfloor} y = \alpha \cdot (x \parallel y)$		CM3
$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$		CM4
$\epsilon \mid x = \delta$		CM5E
$x \mid \epsilon = \delta$		CM6E
$a \cdot x \mid b \cdot y = \gamma(a, b) \cdot (x \parallel y)$		CM7
$(x + y) \mid z = x \mid z + y \mid z$		CM8
$x \mid (y + z) = x \mid y + x \mid z$		CM9
$\partial_H(\epsilon) = \epsilon$		D0
$\partial_H(\alpha) = \alpha$	if $\alpha \notin H$	D1
$\partial_H(\alpha) = \delta$	if $\alpha \in H$	D2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3
$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$		D4
$\tau_I(\epsilon) = \epsilon$		T0
$\tau_I(\alpha) = \alpha$	if $\alpha \notin I$	T1
$\tau_I(\alpha) = \tau$	if $\alpha \in I$	T2
$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$		T3
$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$		T4
$\alpha \cdot (\tau \cdot (x + y) + x) = \alpha \cdot (x + y)$		BE

In the sequel, the notation $\sum_{i=1}^{n} t_i$, where $n \geq 1$, will be used for right-nested alternative compositions. For each $n \in \mathbb{N}^+$,[2] the term $\sum_{i=1}^{n} t_i$ is defined by induction on n as follows: $\sum_{i=1}^{1} t_i = t_1$ and $\sum_{i=1}^{n+1} t_i = t_1 + \sum_{i=1}^{n} t_{i+1}$. In addition, the convention will be used that $\sum_{i=1}^{0} t_i = \delta$.

[2] We write $\mathbb{N}^+$ for the set $\{n \in \mathbb{N} \mid n \geq 1\}$ of positive natural numbers.

3 Imperative $\mathbf{ACP}_\epsilon^\tau$

In this section, $\mathrm{ACP}_\epsilon^\tau$-I, imperative $\mathrm{ACP}_\epsilon^\tau$, is presented. This extension of $\mathrm{ACP}_\epsilon^\tau$ has been inspired by [9]. It extends $\mathrm{ACP}_\epsilon^\tau$ with features that are relevant to processes in which data are involved, such as guarded commands (to deal with processes that only take place if some data-dependent condition holds), data parameterized actions (to deal with process interactions with data transfer), and assignment actions (to deal with data that change in the course of a process).

In $\mathrm{ACP}_\epsilon^\tau$-I, it is assumed that the following has been given with respect to data:

- a (single- or many-sorted) signature $\Sigma_{\mathfrak{D}}$ that includes a sort $\mathbf{D}$ of *data* and constants and/or operators with result sort $\mathbf{D}$;
- a minimal algebra $\mathfrak{D}$ of the signature $\Sigma_{\mathfrak{D}}$.

We write $\mathbb{D}$ for the set of all closed terms over the signature $\Sigma_{\mathfrak{D}}$ that are of sort $\mathbf{D}$.

It is also assumed that a finite or countably infinite set $\mathcal{V}$ of *flexible variables* has been given. A flexible variable is a variable whose value may change in the course of a process.[3] Flexible variables are found under the name program variables in imperative programming. An *evaluation map* is a function from $\mathcal{V}$ to $\mathbb{D}$. We write $\mathcal{EM}$ for the set of all evaluation maps.

The algebraic theory $\mathrm{ACP}_\epsilon^\tau$-I has three sorts: the sort $\mathbf{P}$ of *processes*, the sort $\mathbf{C}$ of *conditions*, and the sort $\mathbf{D}$ of *data*.

It is assumed that there are countably infinite sets of variables of sort $\mathbf{C}$ and $\mathbf{D}$ and that the sets of variables of sort $\mathbf{P}$, $\mathbf{C}$, and $\mathbf{D}$ are mutually disjoint and disjoint from $\mathcal{V}$.

$\mathrm{ACP}_\epsilon^\tau$-I has the constants and operators from $\Sigma_{\mathfrak{D}}$ and in addition the following constants to build terms of sort $\mathbf{D}$:

- for each $v \in \mathcal{V}$, the *flexible variable* constant $v : \to \mathbf{D}$.

We write $\mathcal{D}$ for the set of all closed $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{D}$.

Evaluation maps are intended to provide the data values assigned to flexible variables when a $\mathrm{ACP}_\epsilon^\tau$-I term of sort $\mathbf{D}$ is evaluated. However, in order to fit better in an algebraic setting, they provide closed terms over the signature $\Sigma_{\mathfrak{D}}$ that denote those data values instead. The requirement that $\mathfrak{D}$ is a minimal algebra guarantees that each data value can be represented by a closed term.

$\mathrm{ACP}_\epsilon^\tau$-I has the following constants and operators to build terms of sort $\mathbf{C}$:

- the binary *equality* operator $= : \mathbf{D} \times \mathbf{D} \to \mathbf{C}$;
- the *truth* constant $\mathsf{t} : \to \mathbf{C}$;
- the *falsity* constant $\mathsf{f} : \to \mathbf{C}$;
- the unary *negation* operator $\neg : \mathbf{C} \to \mathbf{C}$;
- the binary *conjunction* operator $\wedge : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$;
- the binary *disjunction* operator $\vee : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$;

[3] The term flexible variable is used for this kind of variables in e.g. [40,31].

– the binary *implication* operator $\rightarrow : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$;
– the unary variable-binding *universal quantification* operator $\forall : \mathbf{C} \rightarrow \mathbf{C}$ that binds a variable of sort $\mathbf{D}$;
– the unary variable-binding *existential quantification* operator $\exists : \mathbf{C} \rightarrow \mathbf{C}$ that binds a variable of sort $\mathbf{D}$.

We write $\mathcal{C}$ for the set of all closed $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{C}$.

Each term from $\mathcal{C}$ can be taken as a formula of a first-order language with equality of $\mathfrak{D}$ by taking the flexible variable constants as additional variables of sort $\mathbf{D}$. The flexible variable constants are implicitly taken as additional variables of sort $\mathbf{D}$ wherever the context asks for a formula. In this way, each term from $\mathcal{C}$ can be interpreted as a formula in $\mathfrak{D}$.

$\mathrm{ACP}_\epsilon^\tau$-I has the constants and operators of $\mathrm{ACP}_\epsilon^\tau$ and in addition the following operators to build terms of sort $\mathbf{P}$:

– the binary *guarded command* operator $:\rightarrow : \mathbf{C} \times \mathbf{P} \rightarrow \mathbf{P}$;
– for each $n \in \mathbb{N}$, for each $a \in \mathsf{A}$, the n-ary *data parameterized action* operator $a : \underbrace{\mathbf{D} \times \cdots \times \mathbf{D}}_{n \text{ times}} \rightarrow \mathbf{P}$;
– for each $v \in \mathcal{V}$, a unary *assignment action* operator $v{:=} : \mathbf{D} \rightarrow \mathbf{P}$;
– for each $\sigma \in \mathcal{EM}$, a unary *evaluation* operator $\mathsf{V}_\sigma : \mathbf{P} \rightarrow \mathbf{P}$.

We write $\mathcal{P}$ for the set of all closed $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{P}$.

The formation rules for $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{P}$, $\mathbf{C}$, and $\mathbf{D}$ are the usual ones for the many-sorted case (see e.g. [39,43]) and in addition the following rule:

– if O is a variable-binding operator $O : S_1 \times \ldots \times S_n \rightarrow S$ that binds a variable of sort S', $t_1, \ldots, t_n$ are terms of sorts $S_1, \ldots, S_n$, respectively, and X is a variable of sort S', then $OX(t_1, \ldots, t_n)$ is a term of sort S (cf. [36]).

The same notational conventions are used as before. Infix notation is also used for the additional binary operators. Moreover, the notation $[v := e]$, where $v \in \mathcal{V}$ and e is a $\mathrm{ACP}_\epsilon^\tau$-I term of sort $\mathbf{D}$, is used for the term $v := (e)$.

The notation $\phi \leftrightarrow \psi$, where ϕ and ψ are $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{C}$, is used for the term $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$. The axioms of $\mathrm{ACP}_\epsilon^\tau$-I (given below) include an equation $\phi = \psi$ for each two terms ϕ and ψ from $\mathcal{C}$ for which the formula $\phi \leftrightarrow \psi$ holds in $\mathfrak{D}$.

Let t be a term from $\mathcal{P}$, ϕ be a term from $\mathcal{C}$, $e_1, \ldots, e_n$ and e be terms from $\mathcal{D}$, and a be a basic action from A. Then the additional operators to build terms of sort $\mathbf{P}$ can be explained as follows:

– the term $\phi :\rightarrow t$ denotes the process that behaves as the process denoted by t if condition ϕ holds and as δ otherwise;
– the term $a(e_1, \ldots, e_n)$ denotes the process that performs the data parameterized action $a(e_1, \ldots, e_n)$ and after that terminates successfully;
– the term $[v := e]$ denotes the process that performs the assignment action $[v := e]$, whose intended effect is the assignment of the result of evaluating e to flexible variable v, and after that terminates successfully;

– the term $\mathsf{V}_\sigma(t)$ denotes the process that behaves the same as the process denoted by t except that each subterm of t that belongs to $\mathcal{D}$ is evaluated using the evaluation map σ updated according to the assignment actions that have taken place at the point where the subterm is encountered.

Evaluation operators are a variant of state operators (see e.g. [1]).

The following closed $\mathrm{ACP}_\epsilon^\tau$-I term ($i, j, d \in \mathcal{V}$) is reminiscent of a program that computes the difference between two integers by subtracting the smaller one from the larger one:

$$[d := i] \cdot ((d \geq j) :\rightarrow [d := d - j] + (d < j) :\rightarrow [d := j - d]) \;.$$

That is, the final value of d is the absolute value of the result of subtracting the initial value of i from the initial value of j. An evaluation operator can be used to show that this is the case for given initial values of i and j. For example, consider the case where the initial values of i and j are 11 and 3, respectively. Let σ be an evaluation map such that $\sigma(i) = 11$ and $\sigma(j) = 3$. Then the following equation can be derived from the axioms of $\mathrm{ACP}_\epsilon^\tau$-I given below:

$$\begin{aligned}&\mathsf{V}_\sigma([d := i] \cdot ((d \geq j) :\rightarrow [d := d - j] + (d < j) :\rightarrow [d := j - d]))\\ &\quad = [d := 11] \cdot [d := 8] \;.\end{aligned}$$

This equation shows that in the case where the initial values of i and j are 11 and 3 the final value of d is 8 (which is the absolute value of the result of subtracting 11 from 3).

An evaluation map σ can be extended homomorphically from flexible variables to $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{D}$ and $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{C}$. This extension is denoted by σ as well. Below, we write $\sigma\{e/v\}$ for the evaluation map σ' defined by $\sigma'(v') = \sigma(v')$ if $v' \neq v$ and $\sigma'(v) = e$.

Three subsets of $\mathcal{P}$ are defined:

$$\begin{aligned}\mathcal{A}^{dpa} &= \textstyle\bigcup_{n \in \mathbb{N}^+} \{a(e_1, \ldots, e_n) \mid a \in \mathsf{A} \land e_1, \ldots, e_n \in \mathcal{D}\} \;,\\ \mathcal{A}^{ass} &= \{[v := e] \mid v \in \mathcal{V} \land e \in \mathcal{D}\} \;,\\ \mathcal{A} &= \{a \mid a \in \mathsf{A}\} \cup \mathcal{A}^{dpa} \cup \mathcal{A}^{ass} \;.\end{aligned}$$

In $\mathrm{ACP}_\epsilon^\tau$-I, the elements of $\mathcal{A}$ are the terms from $\mathcal{P}$ that denote the processes that are considered to be atomic. Henceforth, we write $\mathcal{A}_\tau$ for $\mathcal{A} \cup \{\tau\}$ and $\mathcal{A}_{\tau\delta}$ for $\mathcal{A} \cup \{\tau, \delta\}$.

The axioms of $\mathrm{ACP}_\epsilon^\tau$-I are the axioms presented in Table 1, on the understanding that α now stands for an arbitrary term from $\mathcal{A}_{\tau\delta}$ and H and I now stand for arbitrary subsets of $\mathcal{A}$, and in addition the axioms presented in Table 2. In the latter table, ϕ and ψ stand for arbitrary terms from $\mathcal{C}$, $e, e_1, e_2, \ldots,$ and e', $e'_1, e'_2, \ldots$ stand for arbitrary terms from $\mathcal{D}$, v stands for an arbitrary flexible variable from $\mathcal{V}$, σ stands for an arbitrary evaluation map from $\mathcal{EM}$, a, b, and c stand for arbitrary basic actions from A, and α stands for an arbitrary term from $\mathcal{A}_{\tau\delta}$. Axioms GC1–GC12 have been taken from [2] (using a different

Table 2. Additional axioms of $\mathrm{ACP}_{\epsilon}^{\tau}$-I

$e = e'$	if $\mathfrak{D} \models e = e'$	IMP1
$\phi = \psi$	if $\mathfrak{D} \models \phi \leftrightarrow \psi$	IMP2
$\mathsf{t} :\rightarrow x = x$		GC1
$\mathsf{f} :\rightarrow x = \delta$		GC2
$\phi :\rightarrow \delta = \delta$		GC3
$\phi :\rightarrow (x + y) = \phi :\rightarrow x + \phi :\rightarrow y$		GC4
$\phi :\rightarrow x \cdot y = (\phi :\rightarrow x) \cdot y$		GC5
$\phi :\rightarrow (\psi :\rightarrow x) = (\phi \wedge \psi) :\rightarrow x$		GC6
$(\phi \vee \psi) :\rightarrow x = \phi :\rightarrow x + \psi :\rightarrow x$		GC7
$(\phi :\rightarrow x) \mathbin{\|\!\lfloor} y = \phi :\rightarrow (x \mathbin{\|\!\lfloor} y)$		GC8
$(\phi :\rightarrow x) \mid y = \phi :\rightarrow (x \mid y)$		GC9
$x \mid (\phi :\rightarrow y) = \phi :\rightarrow (x \mid y)$		GC10
$\partial_H(\phi :\rightarrow x) = \phi :\rightarrow \partial_H(x)$		GC11
$\tau_I(\phi :\rightarrow x) = \phi :\rightarrow \tau_I(x)$		GC12
$\mathsf{V}_\sigma(\epsilon) = \epsilon$		V0
$\mathsf{V}_\sigma(\tau \cdot x) = \tau \cdot \mathsf{V}_\sigma(x)$		V1
$\mathsf{V}_\sigma(a \cdot x) = a \cdot \mathsf{V}_\sigma(x)$		V2
$\mathsf{V}_\sigma(a(e_1, \ldots, e_n) \cdot x) = a(\sigma(e_1), \ldots, \sigma(e_n)) \cdot \mathsf{V}_\sigma(x)$		V3
$\mathsf{V}_\sigma([v := e] \cdot x) = [v := \sigma(e)] \cdot \mathsf{V}_{\sigma\{\sigma(e)/v\}}(x)$		V4
$\mathsf{V}_\sigma(x + y) = \mathsf{V}_\sigma(x) + \mathsf{V}_\sigma(y)$		V5
$\mathsf{V}_\sigma(\phi :\rightarrow y) = \sigma(\phi) :\rightarrow \mathsf{V}_\sigma(x)$		V6
$a(e_1, \ldots, e_n) \cdot x \mid b(e'_1, \ldots, e'_n) \cdot y =$ $(e_1 = e'_1 \wedge \ldots \wedge e_n = e'_n) :\rightarrow c(e_1, \ldots, e_n) \cdot (x \parallel y)$	if $\gamma(a, b) = c$	CM7Da
$a(e_1, \ldots, e_n) \cdot x \mid b(e'_1, \ldots, e'_m) \cdot y = \delta$	if $\gamma(a, b) = \delta$ or $n \neq m$	CM7Db
$a(e_1, \ldots, e_n) \cdot x \mid \alpha \cdot y = \delta$	if $\alpha \notin \mathcal{A}^{dpa}$	CM7Dc
$\alpha \cdot x \mid a(e_1, \ldots, e_n) \cdot y = \delta$	if $\alpha \notin \mathcal{A}^{dpa}$	CM7Dd
$[v := e] \cdot x \mid \alpha \cdot y = \delta$		CM7De
$\alpha \cdot x \mid [v := e] \cdot y = \delta$		CM7Df
$\alpha \cdot (\phi :\rightarrow \tau \cdot (x + y) + \phi :\rightarrow x) = \alpha \cdot (\phi :\rightarrow (x + y))$		BED

numbering), but with the axioms with occurrences of Hoare's ternary counterpart of the guarded command operator (see below) replaced by simpler axioms. Axioms CM7Da and CM7Db have been inspired by [9]. Axiom BED is axiom BE generalized to the current setting. An equivalent axiomatization is obtained if axiom BED is replaced by the equation $\alpha \cdot (\phi :\rightarrow \tau \cdot x) = \alpha \cdot (\phi :\rightarrow x)$.

Some earlier extensions of ACP include Hoare's ternary counterpart of the binary guarded command operator (see e.g. [2]). This operator can be defined by the equation $x \triangleleft u \triangleright y = u :\rightarrow x + (\neg u) :\rightarrow y$. From this defining equation, it follows that $u :\rightarrow x = x \triangleleft u \triangleright \delta$. In [26], a unary counterpart of the binary guarded command operator is used. This operator can be defined by the equation $\{u\} = u :\rightarrow \epsilon$. From this defining equation, it follows that $u :\rightarrow x = \{u\} \cdot x$ and also that $\{\mathsf{t}\} = \epsilon$ and $\{\mathsf{f}\} = \delta$.

4 $\mathbf{ACP}_{\epsilon}^{\tau}$-I with Recursion

A closed $\mathrm{ACP}_{\epsilon}^{\tau}$-I term of sort $\mathbf{P}$ denotes a process with a finite upper bound to the number of actions that it can perform. Recursion allows the description of processes without a finite upper bound to the number of actions that it can perform.

A *recursive specification* over $\mathrm{ACP}_{\epsilon}^{\tau}$-I is a set $\{X_i = t_i \mid i \in I\}$, where I is a finite set, each X_i is a variable from $\mathcal{X}$, each t_i is a $\mathrm{ACP}_{\epsilon}^{\tau}$-I term of sort $\mathbf{P}$ in which only variables from $\{X_i \mid i \in I\}$ occur, and $X_i \neq X_j$ for all $i, j \in I$ with $i \neq j$. We write $\mathrm{V}(E)$, where E is a recursive specification over $\mathrm{ACP}_{\epsilon}^{\tau}$-I, for the set of all variables that occur in E. Let E be a recursive specification and let $X \in \mathrm{V}(E)$. Then the unique equation $X = t \; \in E$ is called the *recursion equation for X in E*.

Below, recursive specifications over $\mathrm{ACP}_{\epsilon}^{\tau}$-I are introduced in which the right-hand sides of the recursion equations are linear $\mathrm{ACP}_{\epsilon}^{\tau}$-I terms. The set $\mathcal{L}$ of *linear* $\mathrm{ACP}_{\epsilon}^{\tau}$-I *terms* is inductively defined by the following rules:

- $\delta \in \mathcal{L}$;
- if $\phi \in \mathcal{C}$, then $\phi :\rightarrow \epsilon \in \mathcal{L}$;
- if $\phi \in \mathcal{C}$, $\alpha \in \mathcal{A}_\tau$, and $X \in \mathcal{X}$, then $\phi :\rightarrow \alpha \cdot X \in \mathcal{L}$;
- if $t, t' \in \mathcal{L}$, then $t + t' \in \mathcal{L}$.

Let $t \in \mathcal{L}$. Then we refer to the subterms of t that have the form $\phi :\rightarrow \epsilon$ or the form $\phi :\rightarrow \alpha \cdot X$ as the *summands of t*.

Let X be a variable from $\mathcal{X}$ and let t be an $\mathrm{ACP}_{\epsilon}^{\tau}$-I term in which X occurs. Then an occurrence of X in t is *guarded* if t has a subterm of the form $\alpha \cdot t'$ where $\alpha \in \mathcal{A}$ and t' contains this occurrence of X. Notice that an occurrence of a variable in a linear $\mathrm{ACP}_{\epsilon}^{\tau}$-I term may be not guarded.

A *guarded linear recursive specification* over $\mathrm{ACP}_{\epsilon}^{\tau}$-I is a recursive specification $\{X_i = t_i \mid i \in I\}$ over $\mathrm{ACP}_{\epsilon}^{\tau}$-I where each t_i is a linear $\mathrm{ACP}_{\epsilon}^{\tau}$-I term, and there does not exist an infinite sequence $i_0 \; i_1 \; \ldots$ over I such that, for each $k \in \mathbb{N}$, there is an occurrence of $X_{i_{k+1}}$ in t_{i_k} that is not guarded.

A *linearizable recursive specification* over $\mathrm{ACP}_{\epsilon}^{\tau}$-I is a recursive specification $\{X_i = t_i \mid i \in I\}$ over $\mathrm{ACP}_{\epsilon}^{\tau}$-I where each t_i is rewritable to an $\mathrm{ACP}_{\epsilon}^{\tau}$-I term t'_i, using the axioms of $\mathrm{ACP}_{\epsilon}^{\tau}$-I in either direction and the equations in $\{X_j = t_j \mid j \in I \wedge i \neq j\}$ from left to right, such that $\{X_i = t'_i \mid i \in I\}$ is a guarded linear recursive specification over $\mathrm{ACP}_{\epsilon}^{\tau}$-I.

Table 3. Axioms for guarded linear recursion

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

A solution of a guarded linear recursive specification E over $\mathrm{ACP}_\epsilon^\tau$-I in some model of $\mathrm{ACP}_\epsilon^\tau$-I is a set $\{p_X \mid X \in \mathrm{V}(E)\}$ of elements of the carrier of that model such that each equation in E holds if, for all $X \in \mathrm{V}(E)$, X is assigned p_X. In models of $\mathrm{ACP}_\epsilon^\tau$-I, guarded linear recursive specifications have unique solutions. If $\{p_X \mid X \in \mathrm{V}(E)\}$ is the unique solution of a guarded linear recursive specification E in some model of $\mathrm{ACP}_\epsilon^\tau$-I, then, for each $X \in \mathrm{V}(E)$, p_X is called the X-*component* of the unique solution of E.

$\mathrm{ACP}_\epsilon^\tau$-I is extended with guarded linear recursion by adding constants for solutions of guarded linear recursive specifications over $\mathrm{ACP}_\epsilon^\tau$-I and axioms concerning these additional constants. For each guarded linear recursive specification E over $\mathrm{ACP}_\epsilon^\tau$-I and each $X \in \mathrm{V}(E)$, a constant $\langle X|E\rangle$ of sort $\mathbf{P}$, that stands for the X-component of the unique solution of E, is added to the constants of $\mathrm{ACP}_\epsilon^\tau$-I. The equation RDP and the conditional equation RSP given in Table 3 are added to the axioms of $\mathrm{ACP}_\epsilon^\tau$-I. In RDP and RSP, X stands for an arbitrary variable from $\mathcal{X}$, t stands for an arbitrary $\mathrm{ACP}_\epsilon^\tau$-I term of sort $\mathbf{P}$, E stands for an arbitrary guarded linear recursive specification over $\mathrm{ACP}_\epsilon^\tau$-I, and the notation $\langle t|E\rangle$ is used for t with, for all $X \in \mathrm{V}(E)$, all occurrences of X in t replaced by $\langle X|E\rangle$. Side conditions restrict what X, t and E stand for.

We write $\mathrm{ACP}_\epsilon^\tau$-I+REC for the resulting theory. Furthermore, we write $\mathcal{P}_{rec}$ for the set of all closed $\mathrm{ACP}_\epsilon^\tau$-I+REC terms of sort $\mathbf{P}$.

RDP and RSP together postulate that guarded linear recursive specifications over $\mathrm{ACP}_\epsilon^\tau$-I have unique solutions: the equations $\langle X|E\rangle = \langle t|E\rangle$ and the conditional equations $E \to X = \langle X|E\rangle$ for a fixed E express that the constants $\langle X|E\rangle$ make up a solution of E and that this solution is the only one, respectively.

Because conditional equational formulas must be dealt with in $\mathrm{ACP}_\epsilon^\tau$-I+REC, it is understood that conditional equational logic is used in deriving equations from the axioms of $\mathrm{ACP}_\epsilon^\tau$-I+REC. A complete inference system for conditional equational logic can be found in [25].

We write $T \vdash t = t'$, where T is $\mathrm{ACP}_\epsilon^\tau$-I+REC or $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR (an extension of $\mathrm{ACP}_\epsilon^\tau$-I+REC introduced below), to indicate that the equation $t = t'$ is derivable from the axioms of T using a complete inference system for conditional equational logic.

The following closed $\mathrm{ACP}_\epsilon^\tau$-I+REC term is reminiscent of a program that computes the quotient and remainder of dividing two integers by repeated subtraction ($i, j, q, r \in \mathcal{V}$):

$$[q := 0] \cdot [r := i] \cdot \langle Q|E\rangle\ ,$$

where E is the guarded linear recursive specification that consists of the following two equations ($Q, R \in \mathcal{X}$):

$$Q = (r \geq j) :\to [q := q + 1] \cdot R + (r < j) :\to \epsilon \;,$$
$$R = \mathsf{t} :\to [r := r - j] \cdot Q \;.$$

The final values of q and r are the quotient and remainder of dividing the initial value of i by the initial value of j. An evaluation operator can be used to show that this is the case for given initial values of i and j. For example, consider the case where the initial values of i and j are 11 and 3, respectively. Let σ be an evaluation map such that $\sigma(i) = 11$ and $\sigma(j) = 3$. Then the following equation can be derived from the axioms of $\mathrm{ACP}_\epsilon^\tau$-I+REC:

$$\begin{aligned}&\mathsf{V}_\sigma([q := 0] \cdot [r := i] \cdot \langle X|E \rangle)\\ &\;= [q := 0] \cdot [r := 11] \cdot [q := 1] \cdot [r := 8] \cdot [q := 2] \cdot [r := 5] \cdot [q := 3] \cdot [r := 2] \;.\end{aligned}$$

This equation shows that in the case where the initial values of i and j are 11 and 3 the final values of q and r are 3 and 2 (which are the quotient and remainder of dividing 11 by 3).

Below, use will be made of a reachability notion for the variables occurring in a guarded linear recursive specification over $\mathrm{ACP}_\epsilon^\tau$-I.

Let E be a guarded linear recursive specification over $\mathrm{ACP}_\epsilon^\tau$-I and let $X, Y \in \mathrm{V}(E)$. Then Y *is directly reachable from* X *in* E, written $X \overset{E}{\rightsquigarrow} Y$, if Y occurs in the right-hand side of the recursion equation for X in E. We write $\overset{E}{\rightsquigarrow}{}^*$ for the reflexive transitive closure of $\overset{E}{\rightsquigarrow}$.

Processes with one or more cycles of τ actions are not definable by guarded linear recursion alone, but they are definable by combining guarded linear recursion and abstraction. An example is $\tau_{\{a\}}(\langle X|\{X = a \cdot Y + b, Y = a \cdot X + c\}\rangle)$. The semantics of $\mathrm{ACP}_\epsilon^\tau$-I+REC presented in Section 5 identifies this with $b + \tau \cdot (b + c)$. However, the equation $\tau_{\{a\}}(\langle X|\{X = a \cdot Y + b, Y = a \cdot X + c\}\rangle) = b + \tau \cdot (b + c)$ is not derivable from the axioms of $\mathrm{ACP}_\epsilon^\tau$-I+REC. This is remedied by the addition of the equational axiom schema CFAR (Cluster Fair Abstraction Rule) that will be presented below. This axiom schema makes it possible to abstract from a cycle of actions that are turned into the unobservable action τ, by which only the ways out of the cycle remain. The side condition on the equation concerned requires several notions to be made precise.

Let E be a guarded linear recursive specification over $\mathrm{ACP}_\epsilon^\tau$-I, let $C \subseteq \mathrm{V}(E)$, and let $I \subseteq \mathcal{A}$. Then:

- C *is a cluster for* I *in* E if, for each $\mathrm{ACP}_\epsilon^\tau$-I term $\phi :\to \alpha \cdot X$ of sort $\mathbf{P}$ that is a summand of the right-hand side of the recursion equation for some $X' \in C$ in E, $X \in C$ only if $\phi \equiv \mathsf{t}$ and $\alpha \in I \cup \{\tau\}$.[4]
- for each cluster C for I in E, the *exit set of* C *for* I *in* E, written $\mathit{exits}_{I,E}(C)$, is the set of $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{P}$ defined by $t \in \mathit{exits}_{I,E}(C)$ iff t is a summand of the right-hand side of the recursion equation for some $X' \in C$ in E and one of the following holds:

[4] We write $\equiv$ for syntactic equality.

Table 4. Cluster fair abstraction rule

$\tau \cdot \tau_I(\langle X\|E\rangle) = \tau \cdot \tau_I(\sum_{l=1}^{n} \langle t_l\|E\rangle)$
if for some finite conservative cluster C for I in E,
$X \in C$ and $exits_{I,E}(C) = \{t_1, \ldots, t_n\}$

- $t \equiv \phi :\to \alpha \cdot Y$ for some $\phi \in \mathcal{C}$, $\alpha \in \mathcal{A}_\tau$, and $Y \in \mathrm{V}(E)$ such that $\alpha \notin I \cup \{\tau\}$ or $Y \notin C$;
- $t \equiv \phi :\to \epsilon$ for some $\phi \in \mathcal{C}$;

– C *is a conservative cluster for* I *in* E if C is a cluster for I in E and, for each $X \in C$ and $Y \in exits_{I,E}(C)$, $X \overset{E}{\leadsto}{}^{*} Y$.

The cluster fair abstraction rule is presented in Table 4. In this table, X stands for an arbitrary variable from $\mathcal{X}$, E stands for an arbitrary guarded linear recursive specification over ACP$^\tau_\epsilon$-I, I stands for an arbitrary subset of $\mathcal{A}$, and $t_1, t_2, \ldots$ stand for arbitrary ACP$^\tau_\epsilon$-I terms of sort **P**. A side condition restricts what X, E, I, and $t_1, t_2, \ldots$ stand for.

We write ACP$^\tau_\epsilon$-I+REC+CFAR for the theory ACP$^\tau_\epsilon$-I+REC extended with CFAR.

5 Bisimulation Semantics

In this section, a structural operational semantics of ACP$^\tau_\epsilon$-I+REC is presented and a notion of branching bisimulation equivalence for ACP$^\tau_\epsilon$-I+REC based on this structural operational semantics is defined.

The structural operational semantics of ACP$^\tau_\epsilon$-I+REC consists of

– a binary *conditional transition* relation $\xrightarrow{\ell}$ on $\mathcal{P}_{rec}$ for each $\ell \in \mathcal{EM} \times \mathcal{A}_\tau$;
– a unary *successful termination* relation ${}^{\{\sigma\}}{\downarrow}$ on $\mathcal{P}_{rec}$ for each $\sigma \in \mathcal{EM}$.

We write $t \xrightarrow{\{\sigma\}\,\alpha} t'$ instead of $(t, t') \in \xrightarrow{(\sigma,\alpha)}$ and $t\; {}^{\{\sigma\}}{\downarrow}$ instead of $t \in {}^{\{\sigma\}}{\downarrow}$.

The relations from this structural operational semantics describe what the processes denoted by terms from $\mathcal{P}_{rec}$ are capable of doing as follows:

– $t \xrightarrow{\{\sigma\}\,\alpha} t'$: if the data values assigned to the flexible variables are as defined by σ, then the process denoted by t has the potential to make a transition to the process denoted by t' by performing action α;
– $t\; {}^{\{\sigma\}}{\downarrow}$: if the data values assigned to the flexible variables are as defined by σ, then the process denoted by t has the potential to terminate successfully.

The relations from this structural operational semantics of ACP$^\tau_\epsilon$-I+REC are the smallest relations satisfying the rules given in Table 5. In this table, σ and σ' stand for arbitrary evaluation maps from $\mathcal{EM}$, α stands for an arbitrary action from $\mathcal{A}_\tau$, a, b, and c stand for arbitrary actions from A, e, $e_1, e_2, \ldots$, and $e'_1, e'_2, \ldots$ stand for arbitrary terms from $\mathcal{D}$, H and I stand for arbitrary subset

Table 5. Transition rules for $\mathrm{ACP}_{\epsilon}^{\tau}$-I

$$\frac{}{\alpha\xrightarrow{\{\sigma\}\alpha}\epsilon}$$

$$\frac{}{\epsilon\ ^{\{\sigma\}}\downarrow}$$

$$\frac{x\ ^{\{\sigma\}}\downarrow}{x+y\ ^{\{\sigma\}}\downarrow}\quad\frac{y\ ^{\{\sigma\}}\downarrow}{x+y\ ^{\{\sigma\}}\downarrow}\quad\frac{x\xrightarrow{\{\sigma\}\alpha}x'}{x+y\xrightarrow{\{\sigma\}\alpha}x'}\quad\frac{y\xrightarrow{\{\sigma\}\alpha}y'}{x+y\xrightarrow{\{\sigma\}\alpha}y'}$$

$$\frac{x\ ^{\{\sigma\}}\downarrow,\ y\ ^{\{\sigma\}}\downarrow}{x\cdot y\ ^{\{\sigma\}}\downarrow}\quad\frac{x\ ^{\{\sigma\}}\downarrow,\ y\xrightarrow{\{\sigma\}\alpha}y'}{x\cdot y\xrightarrow{\{\sigma\}\alpha}y'}\quad\frac{x\xrightarrow{\{\sigma\}\alpha}x'}{x\cdot y\xrightarrow{\{\sigma\}\alpha}x'\cdot y}$$

$$\frac{x\ ^{\{\sigma\}}\downarrow,\ y\ ^{\{\sigma\}}\downarrow}{x\parallel y\ ^{\{\sigma\}}\downarrow}\quad\frac{x\xrightarrow{\{\sigma\}\alpha}x'}{x\parallel y\xrightarrow{\{\sigma\}\alpha}x'\parallel y}\quad\frac{y\xrightarrow{\{\sigma\}\alpha}y'}{x\parallel y\xrightarrow{\{\sigma\}\alpha}x\parallel y'}$$

$$\frac{x\xrightarrow{\{\sigma\}a}x',\ y\xrightarrow{\{\sigma\}b}y'}{x\parallel y\xrightarrow{\{\sigma\}c}x'\parallel y'}\ \gamma(a,b)=c$$

$$\frac{x\xrightarrow{\{\sigma\}a(e_1,\ldots,e_n)}x',\ y\xrightarrow{\{\sigma\}b(e_1',\ldots,e_n')}y'}{x\parallel y\xrightarrow{\{\sigma\}c(e_1,\ldots,e_n)}x'\parallel y'}\ \gamma(a,b)=c,\ \mathfrak{D}\models\sigma(e_1=e_1'\wedge\ldots\wedge e_n=e_n')$$

$$\frac{x\xrightarrow{\{\sigma\}\alpha}x'}{x\mathbin{⌊\!⌊} y\xrightarrow{\{\sigma\}\alpha}x'\parallel y}$$

$$\frac{x\xrightarrow{\{\sigma\}a}x',\ y\xrightarrow{\{\sigma\}b}y'}{x\mid y\xrightarrow{\{\sigma\}c}x'\parallel y'}\ \gamma(a,b)=c$$

$$\frac{x\xrightarrow{\{\sigma\}a(e_1,\ldots,e_n)}x',\ y\xrightarrow{\{\sigma\}b(e_1',\ldots,e_n')}y'}{x\mid y\xrightarrow{\{\sigma\}c(e_1,\ldots,e_n)}x'\parallel y'}\ \gamma(a,b)=c,\ \mathfrak{D}\models\sigma(e_1=e_1'\wedge\ldots\wedge e_n=e_n')$$

$$\frac{x\ ^{\{\sigma\}}\downarrow}{\partial_H(x)\ ^{\{\sigma\}}\downarrow}\quad\frac{x\xrightarrow{\{\sigma\}\alpha}x'}{\partial_H(x)\xrightarrow{\{\sigma\}\alpha}\partial_H(x')}\ \alpha\notin H$$

$$\frac{x\ ^{\{\sigma\}}\downarrow}{\tau_I(x)\ ^{\{\sigma\}}\downarrow}\quad\frac{x\xrightarrow{\{\sigma\}\alpha}x'}{\tau_I(x)\xrightarrow{\{\sigma\}\alpha}\tau_I(x')}\ \alpha\notin I\quad\frac{x\xrightarrow{\{\sigma\}\alpha}x'}{\tau_I(x)\xrightarrow{\{\sigma\}\tau}\tau_I(x')}\ \alpha\in I$$

$$\frac{x\ ^{\{\sigma\}}\downarrow}{\phi:\rightarrow x\ ^{\{\sigma\}}\downarrow}\ \mathfrak{D}\models\sigma(\phi)\quad\frac{x\xrightarrow{\{\sigma\}\alpha}x'}{\phi:\rightarrow x\xrightarrow{\{\sigma\}\alpha}x'}\ \mathfrak{D}\models\sigma(\phi)$$

$$\frac{x\ ^{\{\sigma\}}\downarrow}{\mathsf{V}_\sigma(x)\ ^{\{\sigma'\}}\downarrow}\quad\frac{x\xrightarrow{\{\sigma\}\tau}x'}{\mathsf{V}_\sigma(x)\xrightarrow{\{\sigma'\}\tau}\mathsf{V}_\sigma(x')}\quad\frac{x\xrightarrow{\{\sigma\}a}x'}{\mathsf{V}_\sigma(x)\xrightarrow{\{\sigma'\}a}\mathsf{V}_\sigma(x')}$$

$$\frac{x\xrightarrow{\{\sigma\}a(e_1,\ldots,e_n)}x'}{\mathsf{V}_\sigma(x)\xrightarrow{\{\sigma'\}a(\sigma(e_1),\ldots,\sigma(e_n))}\mathsf{V}_\sigma(x')}\quad\frac{x\xrightarrow{\{\sigma\}[v:=e]}x'}{\mathsf{V}_\sigma(x)\xrightarrow{\{\sigma'\}[v:=\sigma(e)]}\mathsf{V}_{\sigma\{\sigma(e)/v\}}(x')}$$

$$\frac{\langle t|E\rangle\ ^{\{\sigma\}}\downarrow}{\langle X|E\rangle\ ^{\{\sigma\}}\downarrow}\ X{=}t\in E\quad\frac{\langle t|E\rangle\xrightarrow{\{\sigma\}\alpha}x'}{\langle X|E\rangle\xrightarrow{\{\sigma\}\alpha}x'}\ X{=}t\in E$$

of $\mathcal{A}$, ϕ stands for an arbitrary term from $\mathcal{C}$, v stands for an arbitrary flexible variable from $\mathcal{V}$, X stands for an arbitrary variable from $\mathcal{X}$, t stands for an arbitrary $\mathrm{ACP}_\epsilon^\tau$-I term of sort $\mathbf{P}$, and E stands for an arbitrary guarded linear recursive specification over $\mathrm{ACP}_\epsilon^\tau$-I.

Two processes are considered equal if they can simulate each other insofar as their observable potentials to make transitions and to terminate successfully are concerned, taking into account the assigments of data values to flexible variables under which the potentials are available. This can be dealt with by means of the notion of branching bisimulation equivalence introduced in [24] adapted to the conditionality of transitions in which the unobservable action τ is performed.

An equivalence relation on the set $\mathcal{A}_\tau$ is needed. Two actions $\alpha, \alpha' \in \mathcal{A}_\tau$ are *data equivalent*, written $\alpha \simeq \alpha'$, iff one of the following holds:

- there exists an $a \in \mathsf{A}_\tau$ such that $\alpha = a$ and $\alpha' = a$;
- for some $n \in \mathbb{N}^+$, there exist an $a \in \mathsf{A}$ and $e_1, \ldots, e_n, e'_1, \ldots, e'_n \in \mathcal{D}$ such that $\mathfrak{D} \models e_1 = e'_1, \ldots, \mathfrak{D} \models e_n = e'_n$, $\alpha = a(e_1, \ldots, e_n)$, and $\alpha' = a(e'_1, \ldots, e'_n)$;
- there exist a $v \in \mathcal{V}$ and $e, e' \in \mathcal{D}$ such that $\mathfrak{D} \models e = e'$, $\alpha = [v := e]$, and $\alpha' = [v := e']$.

We write $[\alpha]$, where $\alpha \in \mathcal{A}_\tau$, for the equivalence class of α with respect to $\simeq$.

For each $\sigma \in \mathcal{EM}$, a binary relation $\xrightarrow{\{\sigma\}}\!\!\!\!\rightarrow$ on $\mathcal{P}_{rec}$ is also needed. It is defined as the smallest binary relation on $\mathcal{P}_{rec}$ such that:

- $t \xrightarrow{\{\sigma\}}\!\!\!\!\rightarrow t$;
- if $t \xrightarrow{\{\sigma\}}\!\!\!\!\rightarrow t'$ and $t' \xrightarrow{\{\sigma\}\,\tau} t''$, then $t \xrightarrow{\{\sigma\}}\!\!\!\!\rightarrow t''$.

Moreover, we write $t \xrightarrow{(\{\sigma\}\,\alpha)} t'$, where $\sigma \in \mathcal{EM}$ and $\alpha \in \mathcal{A}_\tau$, for $t \xrightarrow{\{\sigma\}\,\alpha} t'$ or $\alpha = \tau$ and $t = t'$.

A *branching bisimulation* is a binary relation R on $\mathcal{P}_{rec}$ such that, for all terms $t_1, t_2 \in \mathcal{P}_{rec}$ with $(t_1, t_2) \in R$, the following *transfer conditions* hold:

- if $t_1 \xrightarrow{\{\sigma\}\,\alpha} t'_1$, then there exist an $\alpha' \in [\alpha]$ and $t'_2, t^*_2 \in \mathcal{P}_{rec}$ such that $t_2 \xrightarrow{\{\sigma\}}\!\!\!\!\rightarrow t^*_2 \xrightarrow{(\{\sigma\}\,\alpha')} t'_2$, $(t_1, t^*_2) \in R$, and $(t'_1, t'_2) \in R$;
- if $t_2 \xrightarrow{\{\sigma\}\,\alpha} t'_2$, then there exist an $\alpha' \in [\alpha]$ and $t'_1, t^*_1 \in \mathcal{P}_{rec}$ such that $t_1 \xrightarrow{\{\sigma\}}\!\!\!\!\rightarrow t^*_1 \xrightarrow{(\{\sigma\}\,\alpha')} t'_1$, $(t^*_1, t_2) \in R$, and $(t'_1, t'_2) \in R$;
- if $t_1 \,{}^{\{\sigma\}}\!\!\downarrow$, then there exists a $t^*_2 \in \mathcal{P}_{rec}$ such that $t_2 \xrightarrow{\{\sigma\}}\!\!\!\!\rightarrow t^*_2 \,{}^{\{\sigma\}}\!\!\downarrow$ and $(t_1, t^*_2) \in R$;
- if $t_2 \,{}^{\{\sigma\}}\!\!\downarrow$, then there exists a $t^*_1 \in \mathcal{P}_{rec}$ such that $t_1 \xrightarrow{\{\sigma\}}\!\!\!\!\rightarrow t^*_1 \,{}^{\{\sigma\}}\!\!\downarrow$ and $(t^*_1, t_2) \in R$.

If R is a branching bisimulation, then a pair (t_1, t_2) is said to satisfy the *root condition* in R if the following conditions hold:

- if $t_1 \xrightarrow{\{\sigma\}\,\alpha} t'_1$, then there exist an $\alpha' \in [\alpha]$ and a $t'_2 \in \mathcal{P}_{rec}$ such that $t_2 \xrightarrow{\{\sigma\}\,\alpha'} t'_2$ and $(t'_1, t'_2) \in R$;

– if $t_2 \xrightarrow{\{\sigma\}\,\alpha} t'_2$, then there exist an $\alpha' \in [\alpha]$ and a $t'_1 \in \mathcal{P}_{rec}$ such that $t_1 \xrightarrow{\{\sigma\}\,\alpha'} t'_1$ and $(t'_1, t'_2) \in R$;
– $t_1 \,{}^{\{\sigma\}}{\downarrow}$ iff $t_2 \,{}^{\{\sigma\}}{\downarrow}$.

Two terms $t_1, t_2 \in \mathcal{P}_{rec}$ are *rooted branching bisimulation equivalent*, written $t_1 \leftrightarrow_{\mathsf{rb}} t_2$, if there exists a branching bisimulation R such that $(t_1, t_2) \in R$ and (t_1, t_2) satisfies the root condition in R.

6 Interlude

In the preceding sections, two relatively uncommon choices have been made:

– the choice to include the rather uncommon evaluation operators in the operators of $\mathrm{ACP}^{\tau}_{\epsilon}$-I+REC;
– the choice for a structural operational semantics of $\mathrm{ACP}^{\tau}_{\epsilon}$-I+REC with a transition relation $\xrightarrow{\ell}$ on $\mathcal{P}_{rec}$ for each $\ell \in \mathcal{EM} \times \mathcal{A}_{\tau}$, while a transition relation $\xrightarrow{\ell}$ on $\mathcal{P}_{rec} \times \mathcal{EM}$ for each $\ell \in \mathcal{A}_{\tau}$ is arguably more common.

In this short section, the reasons for these choices are clarified.

The issues which influenced the above-mentioned choices most are:

– the need for a variant of rooted branching bisimulation equivalence that is a congruence with respect to all operators of $\mathrm{ACP}^{\tau}_{\epsilon}$-I+REC;
– the need for a coarser equivalence in cases where parallel composition, left merge, and communication merge are not involved.

With the chosen kind of transition relations, the first need can be fulfilled with a simple and natural generalization of rooted branching bisimulation equivalence as originally introduced in [24], but with the more common kind of transition relations a less obvious variant of rooted branching bisimulation equivalence, a 'stateless' variant in the terminology of [33], has to be devised. As a consequence, with the chosen kind of transition relations, generalizations of existing proof techniques and proof ideas could be used in establishing the soundness and semi-completeness results presented in Section 7, whereas this would not be the case with the more common kind of transition relations.

The variant of rooted branching bisimulation equivalence referred to at the beginning of the previous paragraph is the equivalence $\leftrightarrow_{\mathsf{rb}}$ introduced at the end of Section 5. In order to be a congruence with respect to parallel composition, left merge, and communication merge, $\leftrightarrow_{\mathsf{rb}}$ identifies two terms from $\mathcal{P}_{rec}$ if the processes denoted by them can simulate each other even in the case where the data values assigned to flexible variables may change after each transition. The second need mentioned above is the need for an equivalence that does not take such changes into account in cases where parallel composition, left merge, and communication merge are not involved. Two terms $t, t' \in \mathcal{P}_{rec}$ are equivalent according to this coarser equivalence iff $\mathsf{V}_{\sigma}(t) \leftrightarrow_{\mathsf{rb}} \mathsf{V}_{\sigma}(t')$ for all $\sigma \in \mathcal{EM}$. This means that the coarser equivalence is covered in the semantics of

ACP$_\epsilon^\tau$-I+REC by the choice to include the evaluation operators in the operators of ACP$_\epsilon^\tau$-I+REC.

We have that, for all $t, t' \in \mathcal{P}_{rec}$, for all $\sigma \in \mathcal{EM}$, ACP$_\epsilon^\tau$-I+REC+CFAR $\vdash$ $\mathsf{V}_\sigma(t) = \mathsf{V}_\sigma(t')$ iff $\mathsf{V}_\sigma(t) \leftrightarrows_{\mathsf{rb}} \mathsf{V}_\sigma(t')$ (see Corollary 1 below). So the axioms of ACP$_\epsilon^\tau$-I+REC+CFAR, which constitute an equational axiomatization of $\leftrightarrows_{\mathsf{rb}}$, are also adequate for equational verification of the coarser equivalence.

In [26], an extension of ACP with the empty process constant, the unary counterpart of the binary guarded command operator, and actions to change a data-state is presented. Evaluation maps can be taken as special cases of data-states. For similar reasons as in the case of ACP$_\epsilon^\tau$-I+REC, there is a need for two equivalences. This is not dealt with by the inclusion of evaluation operators. Instead, in equational reasoning, certain axioms may only be applied to terms in which the parallel composition, left merge, and communication merge operators do not occur.

In an appendix, a structural operational semantics of ACP$_\epsilon^\tau$-I+REC is presented which is reminiscent of a symbolic operational semantics in the sense of [29]. It is a structural operational semantics with a transition relation $\xrightarrow{\ell}$ on $\mathcal{P}_{rec}$ for each $\ell \in \mathcal{C}^{sat} \times \mathcal{A}_\tau$, where $\mathcal{C}^{sat}$ is the set of all terms $\phi \in \mathcal{C}$ for which $\mathfrak{D} \not\models \phi \leftrightarrow \mathsf{f}$. In my opinion, this structural operational semantics is intuitively more appealing than the one presented in Section 5, but the definition of the variant of rooted branching bisimulation equivalence based on it is quite unintelligible.

7 Soundness and Completeness

In this section, soundness and (semi-)completeness results with respect to branching bisimulation equivalence for the axioms of ACP$_\epsilon^\tau$-I+REC+CFAR are presented.

Firstly, rooted branching bisimulation equivalence is an equivalence relation indeed.

Proposition 1 (Equivalence). *The relation $\leftrightarrows_{\mathsf{rb}}$ is an equivalence relation.*

Proof. It must be shown that $\leftrightarrows_{\mathsf{rb}}$ is reflexive, symmetric, and transitive.

Let $t \in \mathcal{P}_{rec}$. Then the identity relation I on $\mathcal{P}_{rec}$ is a branching bisimulation such that $(t,t) \in I$ and (t,t) satisfies the root condition in I. Hence, $t \leftrightarrows_{\mathsf{rb}} t$, which proves that $\leftrightarrows_{\mathsf{rb}}$ is reflexive.

Let $t_1, t_2 \in \mathcal{P}_{rec}$ be such that $t_1 \leftrightarrows_{\mathsf{rb}} t_2$, and let R be a branching bisimulation such that $(t_1, t_2) \in R$ and (t_1, t_2) satisfies the root condition in R. Then R^{-1} is a branching bisimulation such that $(t_2, t_1) \in R^{-1}$ and (t_2, t_1) satisfies the root condition in R^{-1}. Hence, $t_2 \leftrightarrows_{\mathsf{rb}} t_1$, which proves that $\leftrightarrows_{\mathsf{rb}}$ is symmetric.

Let $t_1, t_2, t_3 \in \mathcal{P}_{rec}$ be such that $t_1 \leftrightarrows_{\mathsf{rb}} t_2$ and $t_2 \leftrightarrows_{\mathsf{rb}} t_3$, let R be a branching bisimulation such that $(t_1, t_2) \in R$ and (t_1, t_2) satisfies the root condition in R, and let S be a branching bisimulation such that $(t_2, t_3) \in S$ and (t_2, t_3) satisfies the root condition in S. Then $R \circ S$ is a branching bisimulation such that $(t_1, t_3) \in R \circ S$ and (t_1, t_3) satisfies the root condition in $R \circ S$. That $R \circ S$

is a branching bisimulation is proved in the same way as Proposition 7 in [4]. Hence, $t_1 \leftrightarrow_{\mathsf{rb}} t_3$, which proves that $\leftrightarrow_{\mathsf{rb}}$ is transitive. $\square$

Moreover, branching bisimulation equivalence is a congruence with respect to the operators of $\mathrm{ACP}_\epsilon^\tau$-I+REC of which the result sort and at least one argument sort is $\mathbf{P}$.

Proposition 2 (Congruence). *For all terms $t_1, t_1', t_2, t_2' \in \mathcal{P}_{rec}$ and all terms $\phi \in \mathcal{C}$, $t_1 \leftrightarrow_{\mathsf{rb}} t_2$ and $t_1' \leftrightarrow_{\mathsf{rb}} t_2'$ only if $t_1 + t_1' \leftrightarrow_{\mathsf{rb}} t_2 + t_2'$, $t_1 \cdot t_1' \leftrightarrow_{\mathsf{rb}} t_2 \cdot t_2'$, $t_1 \parallel t_1' \leftrightarrow_{\mathsf{rb}} t_2 \parallel t_2'$, $t_1 \mathbin{⌊\!⌊} t_1' \leftrightarrow_{\mathsf{rb}} t_2 \mathbin{⌊\!⌊} t_2'$, $t_1 \mid t_1' \leftrightarrow_{\mathsf{rb}} t_2 \mid t_2'$, $\partial_H(t_1) \leftrightarrow_{\mathsf{rb}} \partial_H(t_2)$, $\tau_I(t_1) \leftrightarrow_{\mathsf{rb}} \tau_I(t_2)$, $\phi :\rightarrow t_1 \leftrightarrow_{\mathsf{rb}} \phi :\rightarrow t_2$, and $\mathsf{V}_\sigma(t_1) \leftrightarrow_{\mathsf{rb}} \mathsf{V}_\sigma(t_2)$.*

Proof. In [23], a SOS rule format is presented which guarantees that the 'standard' version of branching bisimulation equivalence is a congruence. The format concerned is called the RBB safe format. Below, this format is adapted in order to deal with a set of transition labels that contains a special element $\{\sigma\}\,\tau$ for each $\sigma \in \mathcal{EM}$ instead of a single special element τ and with a slightly different version of branching bisimulation equivalence. A definition of a patience rule is needed that differs from the one given in [23]: a *patience rule* for the ith argument of an n-ary operator f is a path rule of the form

$$\frac{x_i \xrightarrow{\{\sigma\}\,\tau} y}{f(x_1, \ldots, x_n) \xrightarrow{\{\sigma\}\,\tau} f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)}\,,$$

where $\sigma \in \mathcal{EM}$. The RBB safe format is adapted by making the following changes to the definition of the RBB safe format as given in [23]:

- in the two syntactic restrictions of the RBB safe format that concern wild arguments, the phrase "a patience rule" is changed to "a patience rule for each $\sigma \in \mathcal{EM}$";
- in the second syntactic restrictions of the RBB safe format that concern wild arguments, the phrase "the relation $\xrightarrow{\tau}$" is changed to "the relation $\xrightarrow{\{\sigma\}\,\tau}$ for some $\sigma \in \mathcal{EM}$".

It is straightforward to check that the proof of Theorem 3.4 from [23] goes through for the adapted RBB safe format and the version of branching bisimulation equivalence considered in this paper. This means that the proposition holds if the rules in Table 5 are in the adapted RBB safe format with respect to some tame/wild labeling of arguments of operators. It is easy to verify that this is the case with the following tame/wild labeling: both arguments of $+$ are tame, the first argument of $\cdot$ is wild and the second argument of $\cdot$ is tame, both arguments of $\parallel$ are wild, both arguments of $⌊\!⌊$ and $\mid$ are tame, the argument of ∂_H and τ_I is wild, the second argument of $:\rightarrow$ is tame, and the argument of V_σ is wild. $\square$

The axioms of $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR are sound with respect to $\leftrightarrow_{\mathsf{rb}}$ for equations between terms from $\mathcal{P}_{rec}$.

Theorem 1 (Soundness). *For all terms $t, t' \in \mathcal{P}_{rec}$, $t = t'$ is derivable from the axioms of* ACP$^\tau_\epsilon$-I+REC+CFAR *only if* $t \leftrightarrow_{\mathsf{rb}} t'$.

Proof. Because it is a congruence, $\leftrightarrow_{\mathsf{rb}}$ respects the inference rules for equality. Therefore, it is sufficient to prove the validity of all substitution instances of each axiom of ACP$^\tau_\epsilon$-I+REC+CFAR.

Below, we write $csi(eq)$, where eq is an equation between two ACP$^\tau_\epsilon$-I+REC terms of sort $\mathbf{P}$, for the set of all closed substitution instances of eq. Moreover, we write R_{id} for the identity relation on $\mathcal{P}_{rec}$.

For each axiom ax of ACP$^\tau_\epsilon$-I+REC+CFAR, a rooted branching bisimulation R_{ax} witnessing the validity of all closed substitution instances of ax can be constructed as follows:

- if ax is one of the axioms A7, CM2E, CM5E, CM6E, GC2 or an instance of one of the axiom schemas D0, D2, T0, GC3, V0, CM7Db–CM7Df:
$$R_{ax} = \{(t, t') \mid t = t' \in csi(ax)\} \;;$$
- if ax is one of the axioms A1–A6, A8, A9, CM4, CM8–CM9, GC1 or an instance of one of the axiom schemas CM3, CM7, D1, D3, D4, T1–T4, GC4–GC12, V1–V6, CM7Da, RDP:
$$R_{ax} = \{(t, t') \mid t = t' \in csi(ax)\} \cup R_{id} \;;$$
- if ax is CM1E:
$$\begin{aligned} R_{ax} = {} & \{(t, t') \mid t = t' \in csi(ax)\} \\ & \cup \{(t, t') \mid t = t' \in csi(x \parallel y = y \parallel x)\} \cup R_{id} \;; \end{aligned}$$
- if ax is an instance of BE:
$$\begin{aligned} R_{ax} = {} & \{(t, t') \mid t = t' \in csi(ax)\} \\ & \cup \{(t, t') \mid t = t' \in csi(\tau \cdot (x + y) + x = x + y)\} \cup R_{id} \;; \end{aligned}$$
- if ax is an instance of BED: similar;
- if ax is an instance $\tau \cdot \tau_I(\langle X|E\rangle) = \tau \cdot \tau_I(\sum_{l=1}^{n} \langle t_l|E\rangle)$ of CFAR:
$$\begin{aligned} R_{ax} = {} & \{(\tau \cdot \tau_I(\langle X|E\rangle), \tau \cdot \tau_I(\textstyle\sum_{l=1}^{n} \langle t_l|E\rangle))\} \\ & \cup \{(\tau_I(\langle X'|E\rangle), \tau_I(\textstyle\sum_{l=1}^{n} \langle t_l|E\rangle)) \mid X' \in C\} \cup R_{id} \;, \end{aligned}$$
where C is the finite conservative cluster for I in E such that $X \in C$ and $exits_{I,E}(C) = \{t_1, \ldots, t_n\}$;
- if ax is an instance $\{X_i = t_i \mid i \in I\} \rightarrow X_j = \langle X_j|\{X_i = t_i \mid i \in I\}\rangle$ $(j \in I)$ of RSP:
$$\begin{aligned} & R_{ax} \\ & = \{(\theta(X_j), \langle X_j|\{X_i = t_i \mid i \in I\}\rangle) \mid j \in I \wedge \theta \in \Theta \wedge \textstyle\bigwedge_{i \in I} \theta(X_i) \leftrightarrow_{\mathsf{rb}} \theta(t_i)\} \\ & \quad \cup R_{id} \;, \end{aligned}$$
where Θ is the set of all functions from $\mathcal{X}$ to $\mathcal{P}_{rec}$ and $\theta(t)$, where $\theta \in \Theta$ and $t \in \mathcal{P}_{rec}$, stands for t with, for all $X \in \mathcal{X}$, all occurrences of X replaced by $\theta(X)$.

For each equational axiom ax of ACP$^\tau_\epsilon$-I+REC+CFAR, it is straightforward to check that the constructed relation R_{ax} witnesses that, for each closed substitution instance $t = t'$ of ax, $t \mathrel{\underline{\leftrightarrow}}_{\mathsf{rb}} t'$. For each conditional equational axiom ax of ACP$^\tau_\epsilon$-I+REC+CFAR, i.e. for each instance of RSP, it is straightforward to check that the constructed relation R_{ax} witnesses that, for the consequent $t = t'$ of each closed substitution instance of an instance of ax, $t \mathrel{\underline{\leftrightarrow}}_{\mathsf{rb}} t'$ if $s \mathrel{\underline{\leftrightarrow}}_{\mathsf{rb}} s'$ for each equation $s = s'$ in the antecedent of the closed substitution instance concerned. □

The axioms of ACP$^\tau_\epsilon$-I+REC+CFAR are incomplete with respect to $\mathrel{\underline{\leftrightarrow}}_{\mathsf{rb}}$ for equations between terms from $\mathcal{P}_{rec}$ and there is no straightforward way to rectify this. Below two semi-completeness results are presented. The next two lemmas are used in the proofs of those results.

A term $t \in \mathcal{P}_{rec}$ is called *abstraction-free* if no abstraction operator occurs in t. A term $t \in \mathcal{P}_{rec}$ is called *bool-conditional* if, for each $\phi \in \mathcal{C}$ that occurs in t, $\mathfrak{D} \models \phi \leftrightarrow \mathsf{t}$ or $\mathfrak{D} \models \phi \leftrightarrow \mathsf{f}$.

Lemma 1. *For all abstraction-free $t \in \mathcal{P}_{rec}$, there exists a guarded linear recursive specification E and $X \in \mathrm{V}(E)$ such that* ACP$^\tau_\epsilon$-I+REC $\vdash t = \langle X|E\rangle$.

Proof. This is easily proved by structural induction on t. □

The proof of Lemma 1 involves constructions of guarded linear recursive specifications from guarded linear recursive specifications for the operators of ACP$^\tau_\epsilon$-I other than the abstraction operators. For the greater part, the constructions are reminiscent of operations on process graphs defined in Sections 2.7 and 4.5.5 from [3].

Lemma 2. *For all bool-conditional $t \in \mathcal{P}_{rec}$, there exists a guarded linear recursive specification E and $X \in \mathrm{V}(E)$ such that* ACP$^\tau_\epsilon$-I+REC+CFAR $\vdash$ $t = \langle X|E\rangle$.

Proof. This is also proved by structural induction on t. The cases other than the case where t is of the form $\tau_I(t')$ are as in the proof of Lemma 1. The case where t is of the form $\tau_I(t')$ is the difficult one. It is proved in the same way as it is done for ACP$^\tau$+REC+CFAR in the proof of Theorem 5.6.2 from [22]. □

The difficult case of the proof of Lemma 2 is the only case in which an application of CFAR is involved.

The following two theorems are the semi-completeness results referred to above.

Theorem 2 (Semi-completeness I). *For all abstraction-free $t, t' \in \mathcal{P}_{rec}$,* ACP$^\tau_\epsilon$-I+REC $\vdash t = t'$ *if* $t \mathrel{\underline{\leftrightarrow}}_{\mathsf{rb}} t'$.

Proof. Because of Lemma 1, Theorem 1, and Proposition 1, it suffices to prove that, for all guarded linear recursive specifications E and E' with $X \in \mathrm{V}(E)$ and $X' \in \mathrm{V}(E')$, ACP$^\tau_\epsilon$-I+REC $\vdash \langle X|E\rangle = \langle X'|E'\rangle$ if $\langle X|E\rangle \mathrel{\underline{\leftrightarrow}}_{\mathsf{rb}} \langle X'|E'\rangle$. This is proved in the same way as it is done for ACP$^\tau$+REC in the proof of Theorem 5.3.2 from [22]. □

Theorem 3 (Semi-completeness II). *For all bool-conditional $t, t' \in \mathcal{P}_{rec}$, $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR $\vdash t = t'$ if $t \leftrightarrows_{\mathsf{rb}} t'$.*

Proof. Because of Lemma 2, Theorem 1, and Proposition 1, it suffices to prove that, for all guarded linear recursive specifications E and E' with $X \in \mathrm{V}(E)$ and $X' \in \mathrm{V}(E')$, $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR $\vdash \langle X|E\rangle = \langle X'|E'\rangle$ if $\langle X|E\rangle \leftrightarrows_{\mathsf{rb}} \langle X'|E'\rangle$. This is proved in the same way as it is done for ACP^τ+REC in the proof of Theorem 5.3.2 from [22]. □

It is due to sufficiently similar shapes of linear $\mathrm{ACP}_\epsilon^\tau$-I terms and linear ACP^τ terms that parts of the proof of Theorems 2 and 3 go in the same way as parts of proofs from [22]. It needs mentioning here that, the body of the proof of Theorem 5.3.2 from [22] is restricted to constants $\langle X|E\rangle$ where E does not contain equations $Y = \tau + \ldots + \tau$ with $Y \not\equiv X$. The corresponding part of the proof of Theorems 2 and 3 is likewise restricted to constants $\langle X|E\rangle$ where E does not contain equations $Y = \phi_1 :\to \tau + \ldots + \phi_n :\to \tau$ with $Y \not\equiv X$. This is not because such an equation can be eliminated, but because it can be replaced by $Y = \phi_1 \vee \ldots \vee \phi_n :\to \epsilon$.

The following is a corollary of Theorems 1 and 3.

Corollary 1. *For all $t, t' \in \mathcal{P}_{rec}$, for all $\sigma \in \mathcal{EM}$, $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR $\vdash$ $\mathsf{V}_\sigma(t) = \mathsf{V}_\sigma(t')$ iff $\mathsf{V}_\sigma(t) \leftrightarrows_{\mathsf{rb}} \mathsf{V}_\sigma(t')$.*

8 Information-Flow Security

In this section, it will be explained how $\mathrm{ACP}_\epsilon^\tau$-I can be used for information-flow security analysis of the kind that is concerned with the leakage of confidential data. However, first, a general idea is given of what information-flow security is about and what results has been produced by research on this subject.

Consider a program whose variables are partitioned into high-security variables and low-security variables. High-security variables are considered to contain confidential data and low-security variables are considered to contain non-confidential data. The information flow in the program is called secure if information derivable from data contained in the high-security variables cannot be inferred from data contained in the low-security variables. Secure information flow means that no confidential data is leaked. A well-known program property that guarantees secure information flow is non-interference. In the case where the program is a deterministic sequential program, non-interference is the property that the data initially contained in high security variables has no effect on the data finally contained in low security variables.

Theoretical work on information-flow security is already done for more than 40 years (see e.g. [5,17,18,14,15]). A great part of the work done until now has been done in a programming-language setting. This work has among other things led to security-type systems for programming languages. The languages concerned vary from languages supporting sequential programming to languages supporting concurrent programming and from languages for programming transformational systems to languages for programming reactive systems (see e.g. [42,41,12,35,10]).

However, work on information-flow security has also been done in a process-algebra setting. In such a setting, the information flow in a process is generally called secure if information derivable from confidential actions cannot be inferred from non-confidential actions (see e.g. [20,37,11,32]). So, in a process-algebra setting, secure information flow usually means that no confidential action is revealed. Moreover, in such a setting, non-interference is the property that the confidential actions has no effect on the non-confidential actions. Recently, work done on information-flow security in a process-algebra setting occasionally deals with the data-oriented notion of secure information flow, but on such occasions program variables are always mimicked by processes (see e.g. [21,30]). $\mathrm{ACP}_{\epsilon}^{\tau}$-I obviates the need to mimic program variables.

In the rest of this section, the interest is in processes that are carried out by systems that have a state comprising a number of data-containing components whose content can be looked up and changed. Moreover, the attention is focussed on processes, not necessarily arising from the execution of a program, in which (a) confidential and non-confidential data contained in the state components of the system in question are looked up and changed and (b) an ongoing interaction with the environment of the system in question is maintained where data are communicated in either direction. In the terminology of $\mathrm{ACP}_{\epsilon}^{\tau}$-I, the state components are called flexible variables. From now on, processes of the kind described above are referred to as processes of the type of interest. The processes that are carried out by many contemporary systems are covered by the processes of the type of interest.

The point of view is taken that the information flow in a process of the type of interest is secure if information derivable from the confidential data contained in state components cannot be inferred from its interaction with the environment. A process property that guarantees secure information flow in this sense is the property that the confidential data contained in state components has no effect on the interaction with the environment. This property, which will be made more precise below, is called the DNII (Data Non-Interference with Interactions) property. For a process with this property, differences in the confidential data contained in state components cannot be observed in (a) what remains of the process in the case where only the actions that are performed to interact with the environment are visible and (b) consequently in the data communicated with the environment.

For each closed $\mathrm{ACP}_{\epsilon}^{\tau}$-I+REC term P of sort $\mathbf{P}$ that denotes a process of the type of interest, it is assumed that a set $\mathit{Low}^P \subseteq \mathcal{V}$ of *low-security* flexible variables of P and a set $\mathit{Ext}^P \subseteq \mathcal{A} \setminus \mathcal{A}^{ass}$ of *external* actions of P have been given. For each closed $\mathrm{ACP}_{\epsilon}^{\tau}$-I+REC term P of sort $\mathbf{P}$ that denotes a process of the type of interest, we write High^P for the set $\{v \in \mathcal{V} \mid v \notin \mathit{Low}^P \wedge v \text{ occurs in } P\}$ of *high-security* flexible variables of P, Int^P for the set $\{\alpha \in \mathcal{A} \mid \alpha \notin \mathit{Ext}^P \wedge$

α occurs in $P\}$ of *internal* actions of P, and Enc^P for the set $\{\alpha \in Int^P \mid \exists \beta \in Int^P \bullet \alpha \mid \beta \neq \delta\}$ of *encapsulated* actions of P.

The flexible variables in Low^P are the flexible variables of P that contain non-confidential data and the flexible variables in $High^P$ are the flexible variables of P that contain confidential data. The actions in Ext^P are the actions that are performed by P to interact with the environment and the actions in Int^P are the actions that are performed by P to do something else than to interact with the environment. The actions in Enc^P are the actions in Int^P that are solely performed to take part in interactions between subprocesses. The actions in Int^P are considered to be invisible in the environment. In earlier work based on a purely action-oriented notion of secure information flow, the actions in Ext^P and Int^P are called low-security actions and high-security actions, respectively, or something similar.

For each closed ACP$^\tau_\epsilon$-I+REC term P of sort $\mathbf{P}$ that denotes a process of the type of interest, P has the DNII property iff

$$\tau_{Int^P}(\mathsf{V}_\sigma(\partial_{Enc^P}(P))) = \tau_{Int^P}(\mathsf{V}_{\sigma'}(\partial_{Enc^P}(P)))$$

for all evaluation maps σ and σ' such that $\sigma(v) = \sigma'(v)$ for all $v \in Low^P$.

Notice that, by Corollary 1, we have

$$\begin{aligned}&\tau_{Int^P}(\mathsf{V}_\sigma(\partial_{Enc^P}(P))) = \tau_{Int^P}(\mathsf{V}_{\sigma'}(\partial_{Enc^P}(P)))\\ &\quad \text{iff } \ \tau_{Int^P}(\mathsf{V}_\sigma(\partial_{Enc^P}(P))) \leftrightarrows_{\mathsf{rb}} \tau_{Int^P}(\mathsf{V}_{\sigma'}(\partial_{Enc^P}(P)))\ .\end{aligned}$$

The DNII property is only one of the process properties related to information flow security that can be defined and verified in ACP$^\tau_\epsilon$-I+REC+CFAR. Insofar information flow security of contemporary systems is concerned, it moreover seems to be an essential property. The DNII property is also one of the process properties related to information flow security that cannot be defined naturally in the process algebras used in earlier work on information flow security (cf. [20,37,11,32]). The problem with those process algebras is that state components must be mimicked by processes in them.

The DNII property is reminiscent of the combination of data non-interference and event non-interference as defined in [38], but a comparison is difficult to make because of rather different semantical bases.

9 Concluding Remarks

I have introduced an ACP-based imperative process algebra which distinguishes itself from other imperative process algebras by being the only one with the following three properties: (a) it supports abstraction from actions that are considered not to be visible; (b) a verification of the equivalence of two processes in its semantics is automatically valid in any semantics that is fully abstract with respect to some notion of observable behaviour; (c) it offers the possibility of equational verification of process equivalence due to an equational axiomatization of the equivalence concerned.

The axioms of the presented imperative process algebra are not complete with respect to the equivalence of processes in its semantics. There is no straightforward way to rectify this. However, two semi-completeness results that may be relevant to various applications of this imperative process algebra have been established. One of those results is at least relevant to information-flow security analysis. The finiteness and linearity restrictions on guarded recursive specifications are not needed for the uniqueness of solutions. However, there would be no semi-completeness results without these restrictions.

In this paper, I build on earlier work on ACP. The axioms of $\mathrm{ACP}_\epsilon^\tau$ have been taken from Section 5.3 of [3] and the axioms for the guarded command operator have been taken from [2]. The evaluation operators have been inspired by [8] and the data parameterized action operators have been inspired by [9].

Appendix: Alternative Bisimulation Semantics

In this appendix, an alternative to the structural operational semantics of $\mathrm{ACP}_\epsilon^\tau$-I+REC is presented and a definition of rooted branching bisimulation equivalence for $\mathrm{ACP}_\epsilon^\tau$-I+REC based on this alternative structural operational semantics is given.

We write $\mathcal{C}^{sat}$ for the set of all terms $\phi \in \mathcal{C}$ for which $\mathfrak{D} \not\models \phi \leftrightarrow \mathsf{f}$. As formulas of a first-order language with equality of $\mathfrak{D}$, the terms from $\mathcal{C}^{sat}$ are the formulas that are satisfiable in $\mathfrak{D}$.

The alternative structural operational semantics of $\mathrm{ACP}_\epsilon^\tau$-I+REC consists of

- a binary conditional transition relation $\xrightarrow{\ell}$ on $\mathcal{P}_{rec}$ for each $\ell \in \mathcal{C}^{sat} \times \mathcal{A}_\tau$;
- a unary successful termination relation ${}^{\{\phi\}}{\downarrow}$ on $\mathcal{P}_{rec}$ for each $\phi \in \mathcal{C}^{sat}$.

We write $t \xrightarrow{\{\phi\}\,\alpha} t'$ instead of $(t, t') \in \xrightarrow{(\phi,\alpha)}$ and $t\ {}^{\{\phi\}}{\downarrow}$ instead of $t \in {}^{\{\phi\}}{\downarrow}$.

The relations from this structural operational semantics describe what the processes denoted by terms from $\mathcal{P}_{rec}$ are capable of doing as follows:

- $t \xrightarrow{\{\phi\}\,\alpha} t'$: if condition ϕ holds for the process denoted by t, then this process has the potential to make a transition to the process denoted by t' by performing action α;
- $t\ {}^{\{\phi\}}{\downarrow}$: if condition ϕ holds for the process denoted by t, then this process has the potential to terminate successfully.

The relations from this structural operational semantics of $\mathrm{ACP}_\epsilon^\tau$-I+REC are the smallest relations satisfying the rules given in Table 6. In this table, α stands for an arbitrary action from $\mathcal{A}_\tau$, ϕ and ψ stand for arbitrary terms from $\mathcal{C}^{sat}$, a, b, and c stand for arbitrary basic actions from A, e, $e_1, e_2, \ldots$, and $e'_1, e'_2, \ldots$ stand for arbitrary terms from $\mathcal{D}$, H and I stand for arbitrary subsets of $\mathcal{A}$, σ stands for an arbitrary evaluation map from $\mathcal{EM}$, v stands for an arbitrary flexible variable from $\mathcal{V}$, X stands for an arbitrary variable from $\mathcal{X}$, t stands for an arbitrary $\mathrm{ACP}_\epsilon^\tau$-I term of sort $\mathbf{P}$, and E stands for an arbitrary guarded linear recursive specification over $\mathrm{ACP}_\epsilon^\tau$-I.

Table 6. Transition rules for $\mathrm{ACP}_\epsilon^\tau$-I

$$\frac{}{\alpha \xrightarrow{\{\mathsf{t}\}\,\alpha} \epsilon}$$

$$\frac{}{\epsilon\ {}^{\{\mathsf{t}\}}\!\downarrow}$$

$$\frac{x\ {}^{\{\phi\}}\!\downarrow}{x+y\ {}^{\{\phi\}}\!\downarrow} \qquad \frac{y\ {}^{\{\phi\}}\!\downarrow}{x+y\ {}^{\{\phi\}}\!\downarrow} \qquad \frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{x+y \xrightarrow{\{\phi\}\,\alpha} x'} \qquad \frac{y \xrightarrow{\{\phi\}\,\alpha} y'}{x+y \xrightarrow{\{\phi\}\,\alpha} y'}$$

$$\frac{x\ {}^{\{\phi\}}\!\downarrow,\ y\ {}^{\{\psi\}}\!\downarrow}{x\cdot y\ {}^{\{\phi\wedge\psi\}}\!\downarrow}\ \mathfrak{D} \not\models \phi\wedge\psi \leftrightarrow \mathsf{f} \qquad \frac{x\ {}^{\{\phi\}}\!\downarrow,\ y \xrightarrow{\{\psi\}\,\alpha} y'}{x\cdot y \xrightarrow{\{\phi\wedge\psi\}\,\alpha} y'}\ \mathfrak{D} \not\models \phi\wedge\psi \leftrightarrow \mathsf{f} \qquad \frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{x\cdot y \xrightarrow{\{\phi\}\,\alpha} x'\cdot y}$$

$$\frac{x\ {}^{\{\phi\}}\!\downarrow,\ y\ {}^{\{\psi\}}\!\downarrow}{x\parallel y\ {}^{\{\phi\wedge\psi\}}\!\downarrow}\ \mathfrak{D} \not\models \phi\wedge\psi \leftrightarrow \mathsf{f} \qquad \frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{x\parallel y \xrightarrow{\{\phi\}\,\alpha} x'\parallel y} \qquad \frac{y \xrightarrow{\{\phi\}\,\alpha} y'}{x\parallel y \xrightarrow{\{\phi\}\,\alpha} x\parallel y'}$$

$$\frac{x \xrightarrow{\{\phi\}\,a} x',\ y \xrightarrow{\{\psi\}\,b} y'}{x\parallel y \xrightarrow{\{\phi\wedge\psi\}\,c} x'\parallel y'}\ \gamma(a,b)=c,\ \mathfrak{D} \not\models \phi\wedge\psi \leftrightarrow \mathsf{f}$$

$$\frac{x \xrightarrow{\{\phi\}\,a(e_1,\ldots,e_n)} x',\ y \xrightarrow{\{\psi\}\,b(e_1',\ldots,e_n')} y'}{x\parallel y \xrightarrow{\{\phi\wedge\psi\wedge e_1=e_1'\wedge\ldots\wedge e_n=e_n'\}\,c(e_1,\ldots,e_n)} x'\parallel y'}\ \begin{array}{c}\gamma(a,b)=c,\\ \mathfrak{D} \not\models \phi\wedge\psi\wedge e_1=e_1'\wedge\ldots\wedge e_n=e_n' \leftrightarrow \mathsf{f}\end{array}$$

$$\frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{x \mathbin{⫡} y \xrightarrow{\{\phi\}\,\alpha} x'\parallel y}$$

$$\frac{x \xrightarrow{\{\phi\}\,a} x',\ y \xrightarrow{\{\psi\}\,b} y'}{x\mid y \xrightarrow{\{\phi\wedge\psi\}\,c} x'\parallel y'}\ \gamma(a,b)=c,\ \mathfrak{D} \not\models \phi\wedge\psi \leftrightarrow \mathsf{f}$$

$$\frac{x \xrightarrow{\{\phi\}\,a(e_1,\ldots,e_n)} x',\ y \xrightarrow{\{\psi\}\,b(e_1',\ldots,e_n')} y'}{x\mid y \xrightarrow{\{\phi\wedge\psi\wedge e_1=e_1'\wedge\ldots\wedge e_n=e_n'\}\,c(e_1,\ldots,e_n)} x'\parallel y'}\ \begin{array}{c}\gamma(a,b)=c,\\ \mathfrak{D} \not\models \phi\wedge\psi\wedge e_1=e_1'\wedge\ldots\wedge e_n=e_n' \leftrightarrow \mathsf{f}\end{array}$$

$$\frac{x\ {}^{\{\phi\}}\!\downarrow}{\partial_H(x)\ {}^{\{\phi\}}\!\downarrow} \qquad \frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{\partial_H(x) \xrightarrow{\{\phi\}\,\alpha} \partial_H(x')}\ \alpha\notin H$$

$$\frac{x\ {}^{\{\phi\}}\!\downarrow}{\tau_I(x)\ {}^{\{\phi\}}\!\downarrow} \qquad \frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{\tau_I(x) \xrightarrow{\{\phi\}\,\alpha} \tau_I(x')}\ \alpha\notin I \qquad \frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{\tau_I(x) \xrightarrow{\{\phi\}\,\tau} \tau_I(x')}\ \alpha\in I$$

$$\frac{x\ {}^{\{\phi\}}\!\downarrow}{\psi :\to x\ {}^{\{\phi\wedge\psi\}}\!\downarrow}\ \mathfrak{D} \not\models \phi\wedge\psi \leftrightarrow \mathsf{f} \qquad \frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{\psi :\to x \xrightarrow{\{\phi\wedge\psi\}\,\alpha} x'}\ \mathfrak{D} \not\models \phi\wedge\psi \leftrightarrow \mathsf{f}$$

$$\frac{x\ {}^{\{\phi\}}\!\downarrow}{\mathsf{V}_\sigma(x)\ {}^{\{\sigma(\phi)\}}\!\downarrow} \qquad \frac{x \xrightarrow{\{\phi\}\,\tau} x'}{\mathsf{V}_\sigma(x) \xrightarrow{\{\sigma(\phi)\}\,\tau} \mathsf{V}_\sigma(x')} \qquad \frac{x \xrightarrow{\{\phi\}\,a} x'}{\mathsf{V}_\sigma(x) \xrightarrow{\{\sigma(\phi)\}\,a} \mathsf{V}_\sigma(x')}$$

$$\frac{x \xrightarrow{\{\phi\}\,a(e_1,\ldots,e_n)} x'}{\mathsf{V}_\sigma(x) \xrightarrow{\{\sigma(\phi)\}\,a(\sigma(e_1),\ldots,\sigma(e_n))} \mathsf{V}_\sigma(x')} \qquad \frac{x \xrightarrow{\{\phi\}\,[v:=e]} x'}{\mathsf{V}_\sigma(x) \xrightarrow{\{\sigma(\phi)\}\,[v:=\sigma(e)]} \mathsf{V}_{\sigma\{\sigma(e)/v\}}(x')}$$

$$\frac{\langle t|E\rangle\ {}^{\{\phi\}}\!\downarrow}{\langle X|E\rangle\ {}^{\{\phi\}}\!\downarrow}\ X{=}t\ \in\ E \qquad \frac{\langle t|E\rangle \xrightarrow{\{\phi\}\,\alpha} x'}{\langle X|E\rangle \xrightarrow{\{\phi\}\,\alpha} x'}\ X{=}t\ \in\ E$$

The alternative structural operational semantics is such that the structural operational semantics presented in Section 5 can be obtained by replacing each transition $t \xrightarrow{\{\phi\}\,\alpha} t'$ by a transition $t \xrightarrow{\{\sigma\}\,\alpha} t'$ for each $\sigma \in \mathcal{EM}$ for which $\mathfrak{D} \models \sigma(\phi)$, and likewise each $t \,{}^{\{\phi\}}{\downarrow}$.

Two processes are considered equal if they can simulate each other insofar as their observable potentials to make transitions and to terminate successfully are concerned. In the case of the alternative structural operational semantics, there are two issues that together complicate matters:

- simply relating a single transition of one of the processes to a single transition of the other process does not work because a transition of one process may be simulated by a set of transitions of another process;
- simply ignoring all transitions in which the unobservable action τ is performed does not work because the observable potentials to make transitions and to terminate successfully may change by such transitions.

The first issue is illustrated by the processes denoted by $\phi \vee \psi :\rightarrow a$ and $\phi :\rightarrow a + \psi :\rightarrow a$: the only transition of the former process is simulated by the two transitions of the latter process. The second issue is illustrated by the processes denoted by $a + \tau \cdot b$ and $a + b$: by making the transition in which the unobservable action τ is performed, the former process loses the potential to make the transition in which the action a is performed before anything has been observed, whereas this potential is an observable potential of the latter process so long as nothing has been observed.

The first issue alone can be dealt with by means of the notion of splitting bisimulation equivalence introduced in [8] and the second issue alone can be dealt with by means of the notion of branching bisimulation equivalence introduced in [24] adapted to the conditionality of transitions in which the unobservable action τ is performed. In order to deal with both issues, the two notions are combined.

We write $t \xrightarrow{(\{\phi\}\,\alpha)} t'$, where $\phi \in \mathcal{C}^{sat}$ and $\alpha \in \mathcal{A}_\tau$, for $t \xrightarrow{\{\phi\}\,\alpha} t'$ or $\alpha = \tau$, $t = t'$, and $\mathfrak{D} \models \phi \leftrightarrow \mathsf{t}$.

The notation $\bigvee \Phi$, where $\Phi = \{\phi_1, \ldots, \phi_n\}$ and $\phi_1, \ldots, \phi_n$ are ACP$^\tau_\epsilon$-I terms of sort $\mathbf{C}$, is used for the ACP$^\tau_\epsilon$-I term $\phi_1 \vee \ldots \vee \phi_n$.

An *ab-bisimulation* is a binary relation R on $\mathcal{P}_{rec}$ such that, for all terms $t_1, t_2 \in \mathcal{P}_{rec}$ with $(t_1, t_2) \in R$, the following transfer conditions hold:

- if $t_1 \xrightarrow{\{\phi\}\,\alpha} t_1'$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \rightarrow \bigvee \Psi$ and, for all $\psi \in \Psi$, there exists an $\alpha' \in [\alpha]$ and, for some $n \in \mathbb{N}$, there exist $t_2^0, \ldots, t_2^n, t_2' \in \mathcal{P}_{rec}$ and $\psi^1, \ldots, \psi^n, \psi' \in \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \psi \leftrightarrow \psi' \wedge \psi^1 \wedge \ldots \wedge \psi^n$, $t_2^0 \equiv t_2$, for all $i \in \mathbb{N}$ with $i < n$, $t_2^i \xrightarrow{\{\psi^{i+1}\}\,\tau} t_2^{i+1}$ and $(t_1, t_2^{i+1}) \in R$, $t_2^n \xrightarrow{(\{\psi'\}\,\alpha')} t_2'$, and $(t_1', t_2') \in R$;
- if $t_2 \xrightarrow{\{\phi\}\,\alpha} t_2'$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \rightarrow \bigvee \Psi$ and, for all $\psi \in \Psi$, there exists an $\alpha' \in [\alpha]$ and, for some $n \in \mathbb{N}$, there exist $t_1^0, \ldots, t_1^n, t_1' \in \mathcal{P}_{rec}$ and $\psi^1, \ldots, \psi^n, \psi' \in \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \psi \leftrightarrow$

$\psi' \wedge \psi^1 \wedge \ldots \wedge \psi^n$, $t_1^0 \equiv t_1$, for all $i \in \mathbb{N}$ with $i < n$, $t_1^i \xrightarrow{\{\psi^{i+1}\}\,\tau} t_1^{i+1}$ and $(t_1^{i+1}, t_2) \in R$, $t_1^n \xrightarrow{(\{\psi'\}\,\alpha')} t_1'$, and $(t_1', t_2') \in R$;

- if $t_1 \;{}^{\{\phi\}}{\downarrow}$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \to \bigvee \Psi$ and, for all $\psi \in \Psi$, for some $n \in \mathbb{N}$, there exist $t_2^0, \ldots, t_2^n \in \mathcal{P}_{rec}$ and $\psi^1, \ldots, \psi^n, \psi' \in \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \psi \leftrightarrow \psi' \wedge \psi^1 \wedge \ldots \wedge \psi^n$, $t_2^0 \equiv t_2$, for all $i \in \mathbb{N}$ with $i < n$, $t_2^i \xrightarrow{\{\psi^{i+1}\}\,\tau} t_2^{i+1}$ and $(t_1, t_2^{i+1}) \in R$, and $t_2^n \;{}^{\{\psi'\}}{\downarrow}$;
- if $t_2 \;{}^{\{\phi\}}{\downarrow}$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \to \bigvee \Psi$ and, for all $\psi \in \Psi$, for some $n \in \mathbb{N}$, there exist $t_1^0, \ldots, t_1^n \in \mathcal{P}_{rec}$ and $\psi^1, \ldots, \psi^n, \psi' \in \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \psi \leftrightarrow \psi' \wedge \psi^1 \wedge \ldots \wedge \psi^n$, $t_1^0 \equiv t_1$, for all $i \in \mathbb{N}$ with $i < n$, $t_1^i \xrightarrow{\{\psi^{i+1}\}\,\tau} t_1^{i+1}$ and $(t_1^{i+1}, t_2) \in R$, and $t_1^n \;{}^{\{\psi'\}}{\downarrow}$.

If R is an ab-bisimulation, then a pair (t_1, t_2) is said to satisfy the root condition in R if the following conditions hold:

- if $t_1 \xrightarrow{\{\phi\}\,\alpha} t_1'$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \to \bigvee \Psi$ and, for all $\psi \in \Psi$, there exist an $\alpha' \in [\alpha]$ and a $t_2' \in \mathcal{P}_{rec}$ such that $t_2 \xrightarrow{\{\psi\}\,\alpha'} t_2'$ and $(t_1', t_2') \in R$;
- if $t_2 \xrightarrow{\{\phi\}\,\alpha} t_2'$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \to \bigvee \Psi$ and, for all $\psi \in \Psi$, there exist an $\alpha' \in [\alpha]$ and a $t_1' \in \mathcal{P}_{rec}$ such that $t_1 \xrightarrow{\{\psi\}\,\alpha'} t_1'$ and $(t_1', t_2') \in R$;
- if $t_1 \;{}^{\{\phi\}}{\downarrow}$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \to \bigvee \Psi$ and, for all $\psi \in \Psi$, $t_2 \;{}^{\{\psi\}}{\downarrow}$;
- if $t_2 \;{}^{\{\phi\}}{\downarrow}$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \to \bigvee \Psi$ and, for all $\psi \in \Psi$, $t_1 \;{}^{\{\psi\}}{\downarrow}$.

Two terms $t_1, t_2 \in \mathcal{P}_{rec}$ are *rooted ab-bisimulation equivalent*, written $t_1 \leftrightarroweq_{\mathsf{rab}} t_2$, if there exists an ab-bisimulation R such that $(t_1, t_2) \in R$ and (t_1, t_2) satisfies the root condition in R.

In the absence of the constant τ, rooted ab-bisimulation equivalence is essentially the same as splitting bisimulation equivalence as defined in [8]. In the absence of all terms of sort $\mathbf{C}$ other than the constants $\mathtt{t}$ and $\mathtt{f}$, rooted ab-bisimulation equivalence is essentially the same as rooted branching bisimulation equivalence as defined in [24].

I conjecture that, for all terms $t_1, t_2 \in \mathcal{P}_{rec}$, $t_1 \leftrightarroweq_{\mathsf{rb}} t_2$ iff $t_1 \leftrightarroweq_{\mathsf{rab}} t_2$.

References

1. Baeten, J.C.M., Bergstra, J.A.: Global renaming operators in concrete process algebra. Information and Control 78(3), 205–245 (1988)
2. Baeten, J.C.M., Bergstra, J.A.: Process algebra with signals and conditions. In: Broy, M. (ed.) Programming and Mathematical Methods. NATO ASI Series, vol. F88, pp. 273–323. Springer-Verlag (1992)
3. Baeten, J.C.M., Weijland, W.P.: Process Algebra, Cambridge Tracts in Theoretical Computer Science, vol. 18. Cambridge University Press, Cambridge (1990)

4. Basten, A.A.: Branching bisimulation is an equivalence indeed! Information Processing Letters 58(3), 141–147 (1996)
5. Bell, D.E., La Padula, L.J.: Secure computer systems: Mathematical foundations and model. Tech. Rep. M74-244, MITRE Corporation (1973)
6. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control 60(1–3), 109–137 (1984)
7. Bergstra, J.A., Middelburg, C.A.: Using Hoare logic in a process algebra setting, to appear in Fundamenta Informaticae, 2021. Preprint: `arXiv:1906.04491v3 [cs.LO]`
8. Bergstra, J.A., Middelburg, C.A.: Splitting bisimulations and retrospective conditions. Information and Computation 204(7), 1083–1138 (2006)
9. Bergstra, J.A., Middelburg, C.A.: A process calculus with finitary comprehended terms. Theory of Computing Systems 53(4), 645–668 (2013)
10. Bohannon, A., et al.: Reactive noninterference. In: CCS '09. pp. 79–90. ACM Press (2009)
11. Bossi, A., Focardi, R., Piazza, C., Rossi, S.: Verifying persistent security properties. Computer Languages, Systems & Structures 30(3–4), 231–258 (2004)
12. Boudol, G., Castellani, I.: Noninterference for concurrent programs and thread systems. Theoretical Computer Science 281(1–2), 109–130 (2002)
13. Bouwman, M.S., Luttik, S.P., Schols, W.R.M., Willemse, T.A.C.: A process algebra with global variables. Electronic Proceedings in Theoretical Computer Science 322, 33–50 (2020)
14. Cohen, E.: Information transmission in computational systems. In: SOSP '77. pp. 133–139. ACM Press (1977)
15. Cohen, E.: Information transmission in sequential programs. In: DeMillo, R.A., Dobkin, D.P., Jones, A.K., Lipton, R.J. (eds.) Foundations of Secure Computation. pp. 297–335. Academic Press (1978)
16. Colvin, R., Hayes, I.J.: CSP with hierarchical state. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. Lecture Notes in Computer Science, vol. 5423, pp. 118–135. Springer-Verlag (2009)
17. Denning, D.E.: A lattice model of secure information flow. Communications of the ACM 19(5), 236–243 (1976)
18. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Communications of the ACM 20(7), 504–513 (1977)
19. Fehnker, A., van Glabbeek, R.J., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: A process algebra for wireless mesh networks. In: Seidl, H. (ed.) ESOP 2012. Lecture Notes in Computer Science, vol. 7211, pp. 295–315. Springer-Verlag (2012)
20. Focardi, R., Gorrieri, R.: A classification of security properties for process algebras. Journal of Computer Security 3(1), 5–33 (1995)
21. Focardi, R., Rossi, S., Sabelfeld, A.: Bridging language-based and process calculi security. In: Sassone, V. (ed.) FOSSACS 2005. Lecture Notes in Computer Science, vol. 3441, pp. 299–315. Springer-Verlag (2005)
22. Fokkink, W.J.: Introduction to Process Algebra. Texts in Theoretical Computer Science, An EATCS Series, Springer-Verlag, Berlin (2000)
23. Fokkink, W.J.: Rooted branching bisimulation as a congruence. Journal of Computer and System Sciences 60(1), 13–37 (2000)
24. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. Journal of the ACM 43(3), 555–600 (1996)
25. Goguen, J.A.: Theorem proving and algebra. `arXiv:2101.02690 [cs.LO]` (January 2021)

26. Groote, J.F., Ponse, A.: Process algebra with guards: Combining Hoare logic with process algebra. Formal Aspects of Computing 6(2), 115–164 (1994)
27. Harel, D., Pnueli, A.: On the development of reactive systems. In: Apt, K. (ed.) Logics and Models of Concurrent Systems. NATO ASI Series, vol. F13, pp. 477–498. Springer-Verlag (1985)
28. Hennessy, M., Ingólfsdóttir, A.: Communicating processes with value-passing and assignments. Formal Aspects of Computing 5(5), 432–466 (1993)
29. Hennessy, M., Lin, H.: Symbolic bisimulations. Theoretical Computer Science 138, 353–389 (1995)
30. Honda, K., Yoshida, N.: A uniform type structure for secure information flow. ACM Transactions on Programming Languages and Systems 29(6), Article 31 (2007)
31. Lamport, L.: The temporal logic of actions. ACM Transactions on Programming Languages and Systems 16(3), 872–923 (1994)
32. Lowe, G.: Semantic models for information flow. Theoretical Computer Science 315(1), 209–256 (2004)
33. Mousavi, M.R., Reniers, M.A., Groote, J.F.: Notions of bisimulation and congruence formats for SOS with data. Information and Computation 200, 107–147 (2005)
34. Nicola, R.D., Pugliese, R.: Testing semantics of asynchronous distributed programs. In: Dam, M. (ed.) LOMAPS 1996. Lecture Notes in Computer Science, vol. 1192, pp. 320–344. Springer-Verlag (1997)
35. O'Neill, K.R., Clarkson, M.R., Chong, S.: Information-flow security for interactive programs. In: CSFW 2006. pp. 190–201. IEEE (2006)
36. Pigozzi, D., Salibra, A.: The abstract variable-binding calculus. Studia Logica 55(1), 129–179 (1995)
37. Ryan, P.Y.A., Schneider, S.A.: Process algebra and non-interference. In: CSFW 1999. pp. 214–227. IEEE (1999)
38. Said, N.B., Cristescu, I.: End-to-end information flow security for web services orchestration. Science of Computer Programming 187, 102376 (2020)
39. Sannella, D., Tarlecki, A.: Algebraic preliminaries. In: Astesiano, E., Kreowski, H.J., Krieg-Brückner, B. (eds.) Algebraic Foundations of Systems Specification, pp. 13–30. Springer-Verlag, Berlin (1999)
40. Schneider, F.B.: On Concurrent Programming. Graduate Texts in Computer Science, Springer-Verlag, Berlin (1997)
41. Smith, G., Volpano, D.: Secure information flow in a multi-threaded imperative language. In: POPL '98. pp. 355–364. ACM Press (1998)
42. Volpano, D., Irvine, C., Smith, G.: A sound type system for secure flow analysis. Journal of Computer Security 4(3), 167–187 (1996)
43. Wirsing, M.: Algebraic specification. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 675–788. Elsevier, Amsterdam (1990)