Scientific Annals of Computer Science vol. 32 (1), 2022, pp. 137–179
doi: 10.7561/SACS.2022.1.137

Imperative Process Algebra with Abstraction

C.A. Middelburg[1]

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 of this kind 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.

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

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

interruption by its environment, output data from input data.[2] 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 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 [33], the term imperative process algebra was coined for process algebras like $\mathrm{ACP}_\epsilon^\tau$-I. Other imperative process algebras are VPLA [27], IPAL [33], 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:

(1) it supports abstraction from actions that are considered not to be visible;

(2) 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. [41]);

(3) it offers the possibility of equational verification of process equivalence.

CSP_σ is the only one of the above-mentioned imperative process algebras that has property (1) and none of them has property (2). $\mathrm{ACP}_\epsilon^\tau$-I is probably unique in being the only imperative process algebra with properties (1), (2) and (3).

Property (1) is achieved by providing a special constant (called the silent step constant), special operators (called abstraction operators), and an appropriate notion of equivalence of processes in the semantics of $\mathrm{ACP}_\epsilon^\tau$-I.

[2] The terms reactive system and transformational system are used here with the meaning given in [26].

Property (2) is achieved by using a notion of branching bisimulation equivalence [41] for the equivalence of processes in the semantics of $\mathrm{ACP}_\epsilon^\tau$-I. Property (3) is achieved by providing an equational axiomatization of the equivalence concerned.

Property (1) is essential for the verification of properties concerning the external behaviour of a system. Property (2) is desirable for such verifications in applications where the final word on what exactly is observable behaviour has not been pronounced. This means that $\mathrm{ACP}_\epsilon^\tau$-I is an interesting process algebra for the verification of properties concerning the external behaviour of a system whose description calls for an imperative process algebra. It makes $\mathrm{ACP}_\epsilon^\tau$-I, among other things, suitable for the verification of properties concerning the information-flow security of a system in which confidential and non-confidential data, contained in state components of the system, are looked up and changed and an ongoing interaction with the environment of the system is maintained.

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, 40, 12, 34, 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 where 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, 36, 11, 31]).

Recent 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, 29]). 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.

The development of $\mathrm{ACP}_\epsilon^\tau$-I was primarily aimed at obtaining an imperative process algebra with the properties that are designated above as essential and desirable for the verification of properties concerning the external behaviour of a system. The starting point of the development of $\mathrm{ACP}_\epsilon^\tau$-I

is $\mathrm{ACP}_\epsilon^\tau$ [3, Section 5.3], which is a non-imperative process algebra with these properties. This makes it a convenient starting point in view of the primary aim of the development.

$\mathrm{ACP}_\epsilon^\tau$-I is closely related to ACP_ϵ^*-D [9]. 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;

(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. However, using the semantics of ACP_ϵ^*-D as presented in [9] as the starting point of the semantics of $\mathrm{ACP}_\epsilon^\tau$-I turned out to result in a semantics that is too complicated to establish the soundness and semi-completeness results.

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. The alternative in question is the above-mentioned result of using the structural operational semantics of ACP_ϵ^*-D as the starting point.

Section 2, Section 3, and the appendix mainly extend the material in Section 2, Section 3, and Section 4, respectively, of [9]. 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 $\mathsf{A} \cup \{\tau\}$.

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 $a : \mathbf{P}$;
- the *silent step* constant $\tau : \mathbf{P}$;
- the *inaction* constant $\delta : \mathbf{P}$;
- the *empty process* constant $\epsilon : \mathbf{P}$;
- the binary *alternative composition* operator $+ : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *sequential composition* operator $\cdot : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *parallel composition* operator $\parallel : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *left merge* operator $\between : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- the binary *communication merge* operator $| : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$;
- for each $H \subseteq \mathsf{A}$ and for $H = \mathsf{A}_\tau$, 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', respectively, let $H \subseteq \mathsf{A}$ or $H = \mathsf{A}_\tau$, and let $I \subseteq \mathsf{A}$. 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 or terminates successfully without performing any 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 — successful termination may take place at any time that both p and p' can terminate successfully;
- $t \between t'$ 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 operators $\between$ and $\mid$ are of an auxiliary nature. They make a finite axiomatization of $\mathrm{ACP}_\epsilon^\tau$ possible.

The operator $\partial_{\mathsf{A}_\tau}$ can also be explained as follows: $\partial_{\mathsf{A}_\tau}(t)$ denotes the process that behaves the same as ϵ if t denotes a process that has the option to behave the same as ϵ and it denotes the process that behaves the same as δ otherwise. In [3, Section 5.3], the symbol $\surd$ is used instead of $\partial_{\mathsf{A}_\tau}$.

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, H stands for an arbitrary subset of A or the set A_τ, and I stands for an arbitrary subset 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.

The occurrence of the strange-looking term $\partial_{\mathsf{A}_\tau}(x) \cdot \partial_{\mathsf{A}_\tau}(y)$ in axiom CM1E deserves some explanation. This term is needed to handle successful termination in the presence of ϵ: it stands for the process that behaves the same as ϵ if both x and y stand for a process that has the option to behave the same as ϵ and it stands for the process that behaves the same as δ otherwise.

Notice that there are no operators ∂_H for $H \subset \mathsf{A}_\tau$ with $\tau \in H$ in $\mathrm{ACP}_\epsilon^\tau$. If one or more of them were present, the equation $\alpha \cdot \delta = \alpha$ would be derivable from the axioms of $\mathrm{ACP}_\epsilon^\tau$.

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 $n \in \mathbb{N}^+$,[3] the term $\sum_{i=1}^n t_i$ is defined by induction on n as follows:

$$\sum_{i=1}^{1} t_i = t_1 \quad \text{and} \quad \sum_{i=1}^{n+1} t_i = t_1 + \sum_{i=1}^{n} t_{i+1} \,.$$

In addition, the convention will be used that $\sum_{i=1}^0 t_i = \delta$.

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

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
$x \parallel y = x \between y + y \between x + x \mid y + \partial_{\mathsf{A}_\tau}(x) \cdot \partial_{\mathsf{A}_\tau}(y)$		CM1E
$\epsilon \between x = \delta$		CM2E
$\alpha \cdot x \between y = \alpha \cdot (x \parallel y)$		CM3
$(x + y) \between z = x \between z + y \between z$		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

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

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 [8]. 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 many-sorted signature $\Sigma_{\mathfrak{D}}$ that includes:
  - a sort $\mathbf{D}$ of *data* and a sort $\mathbf{B}$ of *booleans*;
  - constants of sort $\mathbf{D}$ and/or operators with result sort $\mathbf{D}$;
  - constants $\mathtt{t}$ and $\mathtt{f}$ of sort $\mathbf{B}$ and operators with result sort $\mathbf{B}$;
- a minimal algebra $\mathfrak{D}$ of the signature $\Sigma_{\mathfrak{D}}$ in which the carrier of sort $\mathbf{B}$ has cardinality 2 and the equation $\mathtt{t} = \mathtt{f}$ does not hold.

We write $\mathbb{D}$ for the set of all closed terms over the signature $\Sigma_{\mathfrak{D}}$ that are of sort $\mathbf{D}$. The sort $\mathbf{B}$ is assumed to be given in order to make it possible for operators to serve as predicates.

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.[4] Typical examples of flexible variables are the program variables known from 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 the following sorts: the sort $\mathbf{P}$ of *processes*, the sort $\mathbf{C}$ of *conditions*, and the sorts from $\Sigma_{\mathfrak{D}}$.

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 $\mathcal{V}$.

Below, the constants and operators of $\mathrm{ACP}_\epsilon^\tau$-I are introduced. The operators of $\mathrm{ACP}_\epsilon^\tau$-I include two variable-binding operators. The formation rules for $\mathrm{ACP}_\epsilon^\tau$-I terms are the usual ones for the many-sorted case (see e.g. [38, 43]) and in addition the following rule:

- if O is a variable-binding operator $O : S_1 \times \ldots \times S_n \to S$ 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.

An extensive formal treatment of the phenomenon of variable-binding operators can be found in [35].

[4] The term flexible variable is used for this kind of variables in e.g. [39, 30].

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

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

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 an $\mathrm{ACP}_\epsilon^\tau$-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 $= : \mathbf{B} \times \mathbf{B} \to \mathbf{C}$;
- the binary *equality* operator $= : \mathbf{D} \times \mathbf{D} \to \mathbf{C}$;[5]
- the *truth* constant $\mathtt{t} : \mathbf{C}$;
- the *falsity* constant $\mathtt{f} : \mathbf{C}$;
- 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 $\vee : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$;
- the binary *implication* operator $\Rightarrow : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$;
- the unary variable-binding *universal quantification* operator $\forall : \mathbf{C} \to \mathbf{C}$ that binds a variable of sort $\mathbf{D}$;
- the unary variable-binding *existential quantification* operator $\exists : \mathbf{C} \to \mathbf{C}$ 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

[5] The overloading of $=$ can be trivially resolved if $\Sigma_{\mathfrak{D}}$ is without overloaded symbols.

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

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

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 $\phi{:\!\to} t$ 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 is reminiscent of a program that computes the difference between two integers by subtracting the smaller one from the larger one ($i, j, d \in \mathcal{V}$):

$$[d := i] \cdot ((d \geq j = \mathsf{tt}) :\to [d := d - j] + (d \geq j = \mathsf{ff}) :\to [d := j - d]) \ .^6$$

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:

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

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 **D** and $\mathrm{ACP}_\epsilon^\tau$-I terms of sort **C**. These extensions are 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:

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

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 $\mathcal{A}_{\tau\delta}$, H now stands for an arbitrary subset of $\mathcal{A}$ or the set $\mathcal{A}_\tau$, and I now stands for an arbitrary subset 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, $e_1, e_2, \ldots$,

[6] Here and in examples to come, the carrier of **D** is assumed to be the set of all integers. Moreover, the usual integer constants, operators on integers, and predicates on integers are assumed (where operators with result sort **B** serve as predicates).

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
$\mathsf{t} :\to x = x$		GC1
$\mathsf{f} :\to x = \delta$		GC2
$\phi :\to \delta = \delta$		GC3
$\phi :\to (x + y) = \phi :\to x + \phi :\to y$		GC4
$\phi :\to x \cdot y = (\phi :\to x) \cdot y$		GC5
$\phi :\to (\psi :\to x) = (\phi \wedge \psi) :\to x$		GC6
$(\phi \vee \psi) :\to x = \phi :\to x + \psi :\to x$		GC7
$(\phi :\to x) \between y = \phi :\to (x \between y)$		GC8
$(\phi :\to x) \mid y = \phi :\to (x \mid y)$		GC9
$x \mid (\phi :\to y) = \phi :\to (x \mid y)$		GC10
$\partial_H(\phi :\to x) = \phi :\to \partial_H(x)$		GC11
$\tau_I(\phi :\to x) = \phi :\to \tau_I(x)$		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
$\mathsf{V}_\sigma(\phi :\to y) = \sigma(\phi) :\to \mathsf{V}_\sigma(x)$		V6
$a(e_1, \ldots, e_n) \cdot x \mid b(e'_1, \ldots, e'_n) \cdot y =$ $(e_1 = e'_1 \wedge \ldots \wedge e_n = e'_n) :\to c(e_1, \ldots, e_n) \cdot (x \parallel y)$	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
$\alpha \cdot (\phi :\to \tau \cdot (x + y) + \phi :\to x) = \alpha \cdot (\phi :\to (x + y))$		BED

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 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 [8]. 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 $x \lhd u \rhd y = u :\rightarrow x + (\neg\, u) :\rightarrow y$. From this defining equation, it follows that $u :\rightarrow x = x \lhd u \rhd \delta$. In [25], 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 $\mathrm{ACP}_\epsilon^\tau$-I with Recursion

A closed $\mathrm{ACP}_\epsilon^\tau$-I term of sort **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 **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 $\mathrm{vars}(E)$, 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 $X \in \mathrm{vars}(E)$. 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*.

Table 3: Axioms for guarded linear recursion

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

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. An occurrence of a variable X in a linear $\mathrm{ACP}_\epsilon^\tau$-I term may not be guarded because a linear $\mathrm{ACP}_\epsilon^\tau$-I term may have summands of the form $\phi :\rightarrow \tau \cdot X$.

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.

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 $\{p_X \mid X \in \mathrm{vars}(E)\}$ of elements of the carrier of that model such that each equation in E holds if, for all $X \in \mathrm{vars}(E)$, X is assigned p_X. A guarded linear recursive specification has a unique solution under the equivalence defined in Section 5 for $\mathrm{ACP}_\epsilon^\tau$-I extended with guarded linear recursion. If $\{p_X \mid X \in \mathrm{vars}(E)\}$ is the unique solution of a guarded linear recursive specification E, then, for each $X \in \mathrm{vars}(E)$, 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 $X \in \mathrm{vars}(E)$, 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 (Recursive Definition Principle) and the conditional equation RSP (Recursive Specification Principle) 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

$X \in \mathrm{vars}(E)$, 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 $E \Rightarrow X = \langle X|E\rangle$ 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 for example be found in [3, 24].

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 by repeated subtraction the quotient and remainder of dividing a non-negative integer by a positive integer ($i, j, q, r \in \mathcal{V}$):

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

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

$$\begin{aligned} Q &= (r \geq j = \mathtt{tt}) :\to [q := q + 1] \cdot R + (r \geq j = \mathtt{ff}) :\to \epsilon \;, \\ R &= \mathsf{t} :\to [r := r - j] \cdot Q \;. \end{aligned}$$

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:

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

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 $X, Y \in \mathrm{vars}(E)$. Then Y *is directly reachable from* X in E, written $X \stackrel{E}{\rightsquigarrow} Y$, if Y occurs in the right-hand side of the recursion equation for X in E. We write $\stackrel{E}{\rightsquigarrow}{}^*$ for the reflexive transitive closure of $\stackrel{E}{\rightsquigarrow}$.

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 $C \subseteq \mathrm{vars}(E)$, and let $I \subseteq \mathcal{A}$. Then:

- C *is a cluster for* I *in* E if, for each $\mathrm{ACP}_\epsilon^\tau$-I term $\phi :\rightarrow \alpha \cdot X$ 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 $\phi \equiv \mathtt{t}$ and $\alpha \in I \cup \{\tau\}$;[7]
- 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:
  - $t \equiv \phi :\rightarrow \alpha \cdot Y$ for some $\phi \in \mathcal{C}$, $\alpha \in \mathcal{A}_\tau$, and $Y \in \mathrm{vars}(E)$ such that $\alpha \notin I \cup \{\tau\}$ or $Y \notin C$;
  - $t \equiv \phi :\rightarrow \epsilon$ for some $\phi \in \mathcal{C}$;

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

Table 4: Cluster fair abstraction rule

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

- *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 $Y \in \mathit{exits}_{I,E}(C)$, $X \overset{E}{\rightsquigarrow}{}^{*}\, Y$.

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 $\mathrm{ACP}_\epsilon^\tau$-I, I stands for an arbitrary subset of $\mathcal{A}$, and $t_1, t_2, \ldots$ stand for arbitrary $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{P}$. A side condition restricts what X, E, I, and $t_1, t_2, \ldots$ stand for.

CFAR expresses that every cluster of τ actions will be exited sooner or later. This is a fairness assumption made in the verification of many properties concerning the external behaviour of systems.

We write $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR for the theory $\mathrm{ACP}_\epsilon^\tau$-I+REC extended with CFAR.

5 Bisimulation Semantics

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

The 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 $\ell \in \mathcal{EM} \times \mathcal{A}_\tau$;
- 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 the 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 the structural operational semantics of $\mathrm{ACP}_\epsilon^\tau$-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 stands for an arbitrary subset of $\mathcal{A}$ or the set $\mathcal{A}_\tau$, I stands for an arbitrary subset 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.

The rules in Table 5 have the form $\frac{p_1, \ldots, p_n}{c}\, s$, where s is optional. They are to be read as "if p_1 and $\ldots$ and p_n then c, provided s". As usual, $p_1, \ldots, p_n$ are called the premises and c is called the conclusion. A side condition s, if present, serves to restrict the applicability of a rule. If a rule has no premises, then nothing is displayed above the horizontal bar.

Because the rules in Table 5 constitute an inductive definition, $t \xrightarrow{\{\sigma\}\,\alpha} t'$ or $t\ {}^{\{\sigma\}}\!\!\downarrow$ holds iff it can be inferred from these rules. For instance, for $a, b, c \in \mathsf{A}$, $v, v' \in \mathcal{V}$, and $\sigma \in \mathcal{EM}$ such that $\sigma(v) = \sigma(v')$, we have that $(v = v') :\to (a + b) \cdot c \xrightarrow{\{\sigma\}\,a} \epsilon \cdot c$ can be inferred by applying the first rule, the third rule for $+$, the second rule for $:\to$, and the third rule for $\cdot$ in that order.

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 [41] 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 $\alpha' = a(e'_1, \ldots, e'_n)$;

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

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

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

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

$$\frac{x \ \{\sigma\}\!\downarrow,\ y \ \{\sigma\}\!\downarrow}{x \cdot y \ \{\sigma\}\!\downarrow} \qquad \frac{x \ \{\sigma\}\!\downarrow,\ y \xrightarrow{\{\sigma\}\,\alpha} y'}{x \cdot y \xrightarrow{\{\sigma\}\,\alpha} y'} \qquad \frac{x \xrightarrow{\{\sigma\}\,\alpha} x'}{x \cdot y \xrightarrow{\{\sigma\}\,\alpha} x' \cdot y}$$

$$\frac{x \ \{\sigma\}\!\downarrow,\ y \ \{\sigma\}\!\downarrow}{x \parallel y \ \{\sigma\}\!\downarrow} \qquad \frac{x \xrightarrow{\{\sigma\}\,\alpha} x'}{x \parallel y \xrightarrow{\{\sigma\}\,\alpha} x' \parallel y} \qquad \frac{y \xrightarrow{\{\sigma\}\,\alpha} y'}{x \parallel y \xrightarrow{\{\sigma\}\,\alpha} x \parallel y'}$$

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

$$\frac{x \xrightarrow{\{\sigma\}\,a(e_1,\ldots,e_n)} x',\ y \xrightarrow{\{\sigma\}\,b(e'_1,\ldots,e'_n)} y'}{x \parallel y \xrightarrow{\{\sigma\}\,c(e_1,\ldots,e_n)} x' \parallel y'} \ \gamma(a,b) = c,\ \mathfrak{D} \models \sigma(e_1 = e'_1 \wedge \ldots \wedge e_n = e'_n)$$

$$\frac{x \xrightarrow{\{\sigma\}\,\alpha} x'}{x \between y \xrightarrow{\{\sigma\}\,\alpha} x' \parallel y}$$

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

$$\frac{x \xrightarrow{\{\sigma\}\,a(e_1,\ldots,e_n)} x',\ y \xrightarrow{\{\sigma\}\,b(e'_1,\ldots,e'_n)} y'}{x \mid y \xrightarrow{\{\sigma\}\,c(e_1,\ldots,e_n)} x' \parallel y'} \ \gamma(a,b) = c,\ \mathfrak{D} \models \sigma(e_1 = e'_1 \wedge \ldots \wedge e_n = e'_n)$$

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

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

$$\frac{x \ \{\sigma\}\!\downarrow}{\phi :\to x \ \{\sigma\}\!\downarrow} \ \mathfrak{D} \models \sigma(\phi) \qquad \frac{x \xrightarrow{\{\sigma\}\,\alpha} x'}{\phi :\to x \xrightarrow{\{\sigma\}\,\alpha} x'} \ \mathfrak{D} \models \sigma(\phi)$$

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

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

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

- 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 $\sigma \in \mathcal{EM}$, the binary relation $\xrightarrow{\{\sigma\}}\!\!\!\!\!\rightarrow$ on $\mathcal{P}_{rec}$ defined as the reflexive transitive closure of $\xrightarrow{\{\sigma\}\,\tau}$ is also needed.

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 both $\alpha = \tau$ 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 $t_2 \xrightarrow{\{\sigma\}}\!\!\!\!\!\rightarrow t_2^*$, $t_2^* \xrightarrow{(\{\sigma\}\,\alpha')} t_2'$, $(t_1, t_2^*) \in R$, 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 $t_1 \xrightarrow{\{\sigma\}}\!\!\!\!\!\rightarrow t_1^*$, $t_1^* \xrightarrow{(\{\sigma\}\,\alpha')} t_1'$, $(t_1^*, t_2) \in R$, and $(t_1', t_2') \in R$;
- if $t_1 \,{}^{\{\sigma\}}\!\!\downarrow$, then there exists a $t_2^* \in \mathcal{P}_{rec}$ such that $t_2 \xrightarrow{\{\sigma\}}\!\!\!\!\!\rightarrow t_2^*$, $t_2^* \,{}^{\{\sigma\}}\!\!\downarrow$, and $(t_1, t_2^*) \in R$;
- if $t_2 \,{}^{\{\sigma\}}\!\!\downarrow$, then there exists a $t_1^* \in \mathcal{P}_{rec}$ such that $t_1 \xrightarrow{\{\sigma\}}\!\!\!\!\!\rightarrow t_1^*$, $t_1^* \,{}^{\{\sigma\}}\!\!\downarrow$, 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 $t_1 \leftrightarrows_{\mathsf{rb}} t_2$, 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.

In Section 7, it is proved that $\leftrightarrows_{\mathsf{rb}}$ 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 $\mathbf{P}$. Without the root condition, $\leftrightarrows_{\mathsf{rb}}$ would not be a congruence with respect to the operator $+$. For example, it would be the case that $\tau \cdot a \leftrightarrows_{\mathsf{rb}} a$ and not $\tau \cdot a + b \leftrightarrows_{\mathsf{rb}} a + b$.

Let R be a branching bisimulation such that $(t_1, t_2) \in R$ and the pair (t_1, t_2) satisfies the root condition in R. Then we say that R is a branching bisimulation *witnessing* $t_1 \leftrightarrows_{\mathsf{rb}} t_2$.

6 Interlude

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

- the choice to include the rather unusual evaluation operators, i.e. V_σ for each $\sigma \in \mathcal{EM}$, in the operators of $\mathrm{ACP}_\epsilon^\tau$-I+REC;
- the choice for a structural operational semantics of $\mathrm{ACP}_\epsilon^\tau$-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 $\mathrm{ACP}_\epsilon^\tau$-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 [41], 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 [32], 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 $\leftrightarrows_{\mathsf{rb}}$ introduced at the end of Section 5. In order to be a congruence with respect to parallel composition, left merge, and communication merge, $\leftrightarrows_{\mathsf{rb}}$ 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 — through assignment actions performed by parallel processes. The second need mentioned above is the need for an equivalence that does not takes 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 $\mathsf{V}_\sigma(t) \leftrightarrows_{\mathsf{rb}} \mathsf{V}_\sigma(t')$ for all $\sigma \in \mathcal{EM}$. This means that the coarser equivalence is covered in the semantics of $\mathrm{ACP}_\epsilon^\tau$-I+REC by the choice to include the evaluation operators in the operators of $\mathrm{ACP}_\epsilon^\tau$-I+REC.

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

In [25], 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 $\mathrm{ACP}_\epsilon^\tau$-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 $\mathrm{ACP}_\epsilon^\tau$-I+REC is presented which is reminiscent of a symbolic operational semantics in the sense of [28]. 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 $\mathrm{ACP}_\epsilon^\tau$-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 $R \circ S$.[8] That $R \circ S$ is a branching bisimulation is proved in the same way as Proposition 7 in [4]. Hence, $t_1 \leftrightarrows_{\mathsf{rb}} t_3$, which proves that $\leftrightarrows_{\mathsf{rb}}$ is transitive. □

Moreover, rooted 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 **P**.

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

Proof: A detailed proof would contain an adapted copy of at least ten pages from [23]. Therefore, only an outline of the proof is given here. In order to fully understand the outline, the above-mentioned paper must be consulted.

In [23], an 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]:

[8] We write $R \circ S$ for the composition of R with S.

a *patience rule* for the ith argument of an n-ary operator f is a path rule of the form

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

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 $\|$ are wild, both arguments of $\between$ and $|$ 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. □

The tame/wild labeling given at the end of the proof of Proposition 2 is provided so that the reader who consults [23] can easily check that the rules in Table 5 are in the adapted RBB safe format.

Below, the soundness of the axiom system of $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR with respect to $\leftrightarrows_{\mathsf{rb}}$ for equations between terms from $\mathcal{P}_{rec}$ will be established.

The following terminology will be used in the soundness proof:

- an equation *eq* of $\mathrm{ACP}_\epsilon^\tau$-I+REC terms of sort $\mathbf{P}$ is said to be *valid with respect to* $\leftrightarrows_{\mathsf{rb}}$ if, for each closed substitution instance $t = t'$ of *eq*, $t \leftrightarrows_{\mathsf{rb}} t'$ and
- a conditional equation *ceq* of $\mathrm{ACP}_\epsilon^\tau$-I+REC terms of sort $\mathbf{P}$ is said to be *valid with respect to* $\leftrightarrows_{\mathsf{rb}}$ if, for each closed substitution instance $\{t_i = t_i' \mid i \in I\} \Rightarrow t = t'$ of *ceq*, $t \leftrightarrows_{\mathsf{rb}} t'$ if $t_i \leftrightarrows_{\mathsf{rb}} t_i'$ for each $i \in I$.

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

Proof: Because $\leftrightarrows_{\mathsf{rb}}$ is a congruence with respect to all operators from the signature of $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR, only the validity of each axiom of $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR has to be proved.

Below, we write $csi(eq)$, where eq is an equation of $\mathrm{ACP}_\epsilon^\tau$-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 $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR, a rooted branching bisimulation R_{ax} witnessing the validity 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 $\tau \cdot \tau_I(\langle X|E\rangle) = \tau \cdot \tau_I\left(\sum_{l=1}^{n} \langle t_l|E\rangle\right)$ of CFAR:
$$\begin{aligned} R_{ax} = {} & \left\{\left(\tau \cdot \tau_I(\langle X|E\rangle), \tau \cdot \tau_I\left(\sum_{l=1}^{n} \langle t_l|E\rangle\right)\right)\right\} \\ & \cup \left\{\left(\tau_I(\langle X'|E\rangle), \tau_I\left(\sum_{l=1}^{n} \langle t_l|E\rangle\right)\right) \;\middle|\; X' \in C\right\} \cup R_{id} , \end{aligned}$$

where C is the finite conservative cluster for I in E such that $X \in C$ and $\mathit{exits}_{I,E}(C) = \{t_1, \ldots, t_n\}$;

- 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:

$$R_{ax} = \{(\theta(X_j), \langle X_j | \{X_i = t_i \mid i \in I\}\rangle) \mid j \in I \wedge \theta \in \Theta \wedge \textstyle\bigwedge_{i \in I} \theta(X_i) \leftrightarrows_{\mathsf{rb}} \theta(t_i)\} \cup R_{id} \ ,$$

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 $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR, it is straightforward to check that the constructed relation R_{ax} is a branching bisimulation witnessing, for each closed substitution instance $t = t'$ of ax, $t \leftrightarrows_{\mathsf{rb}} t'$. For each conditional equational axiom ax of $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR, i.e. for each instance of RSP, it is straightforward to check that the constructed relation R_{ax} is a branching bisimulation witnessing, for each closed substitution instance $\{t_i = t'_i \mid i \in I\} \Rightarrow t = t'$ of ax, $t \leftrightarrows_{\mathsf{rb}} t'$ if $t_i \leftrightarrows_{\mathsf{rb}} t'_i$ for each $i \in I$. □

The axioms of $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR are incomplete with respect to $\leftrightarrows_{\mathsf{rb}}$ 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 $X \in \mathrm{vars}(E)$ such that* $\mathrm{ACP}_\epsilon^\tau$-I+REC $\vdash t = \langle X|E\rangle$.

Proof: This is easily proved by structural induction on t. The proof involves constructions of guarded linear recursive specifications from guarded linear recursive specifications for the operators of $\mathrm{ACP}_\epsilon^\tau$-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 $X \in \mathrm{vars}(E)$ such that* $\mathrm{ACP}_\epsilon^\tau$-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^τ+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 $t, t' \in \mathcal{P}_{rec}$,* $\mathrm{ACP}_\epsilon^\tau$-I+REC $\vdash t = t'$ *if* $t \leftrightarrows_{\mathsf{rb}} t'$.

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 $X \in \mathrm{vars}(E)$ and $X' \in \mathrm{vars}(E')$, $\mathrm{ACP}_\epsilon^\tau$-I+REC $\vdash \langle X|E\rangle = \langle X'|E'\rangle$ 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]. □

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 $X \in \mathrm{vars}(E)$ and $X' \in \mathrm{vars}(E')$, $\mathrm{ACP}_\epsilon^\tau$-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 $Y = \phi_1 \vee \ldots \vee \phi_n :\rightarrow \epsilon$.

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 have 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 since the 1970s (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, 40, 12, 34, 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, 36, 11, 31]). 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 have 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, 29]). $\mathrm{ACP}_\epsilon^\tau$-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 the following has been given:

- a set $\mathit{Low}^P \subseteq \mathcal{V}$ of *low-security* flexible variables of P;
- a set $\mathit{Ext}^P \subseteq \mathsf{A} \cup \bigcup_{n \in \mathbb{N}^+} \{a(e_1, \ldots, e_n) \mid a \in \mathsf{A} \wedge e_1, \ldots, e_n \in \mathbb{D}\}$ of *external* actions of P.

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 define the following sets:

$$\begin{aligned} \mathit{High}^P &= \{v \in \mathcal{V}^P \mid v \notin \mathit{Low}^P\} \,, \\ \mathit{Int}^P &= \{\alpha \in \mathcal{A}^P \mid \alpha \notin \mathit{Ext}^P\} \,, \end{aligned}$$

where

$$\begin{aligned} \mathcal{V}^P &= \{v \in \mathcal{V} \mid v \text{ occurs in } P\} \,, \\ \mathcal{A}^P &= \{a \in \mathsf{A} \mid a \text{ occurs in } P\} \\ &\quad \cup \textstyle\bigcup_{n \in \mathbb{N}^+} \{a(\sigma(e_1), \ldots, \sigma(e_n)) \mid \sigma \in \mathcal{EM} \wedge a(e_1, \ldots, e_n) \text{ occurs in } P\} \\ &\quad \cup \{[v := \sigma(e)] \mid \sigma \in \mathcal{EM} \wedge [v := e] \text{ occurs in } P\} \,. \end{aligned}$$

High^P is called the set of *high-security* flexible variables of P and Int^P is called the set of *internal* 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 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 $\mathrm{ACP}_\epsilon^\tau$-I+REC term P of sort $\mathbf{P}$ that denotes a process of the type of interest, P has the DNII property iff

$$\mathrm{ACP}_\epsilon^\tau\text{-I+REC+CFAR} \vdash \tau_{\mathit{Int}^P}(\mathsf{V}_\sigma(P)) = \tau_{\mathit{Int}^P}(\mathsf{V}_{\sigma'}(P))$$

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

This definition is justified by the fact, which follows from Corollary 1, that

$$\begin{aligned} &\mathrm{ACP}_\epsilon^\tau\text{-I+REC+CFAR} \vdash \tau_{\mathit{Int}^P}(\mathsf{V}_\sigma(P)) = \tau_{\mathit{Int}^P}(\mathsf{V}_{\sigma'}(P)) \\ &\qquad \text{iff } \ \tau_{\mathit{Int}^P}(\mathsf{V}_\sigma(P)) \leftrightarrows_{\mathsf{rb}} \tau_{\mathit{Int}^P}(\mathsf{V}_{\sigma'}(P)) \,. \end{aligned}$$

The left-hand side and the right-hand side of the equation in the above definition denote the processes that remain of the process denoted by P in the case that the data values assigned to the flexible variables are initially as defined by σ and σ', respectively, and moreover all internal actions of P are

not visible. The condition imposed on the evaluation maps σ and σ' tells us that the equation must always hold if, for each low-security flexible variable of P, the data values assigned to it according to σ and σ' are the same. This corresponds to the intuitive idea mentioned above that, for a process with the DNII property, differences in the confidential data cannot be observed in what remains of the process in the case where only the actions that are performed to interact with the environment are visible.

Assume that $h, l \in \mathcal{V}$ and $a, b \in \mathsf{A}$. Let P be the closed $\mathrm{ACP}_\epsilon^\tau$-I+REC term

$$(h = 0) :\rightarrow [l := l + 1] \cdot a + \neg(h = 0) :\rightarrow a \cdot [l := l + 1] + b$$

of sort $\mathbf{P}$ with $\mathit{Low}^P = \{l\}$ and $\mathit{Ext}^P = \{a, b\}$. P is a very simple example of a term of which it may not be immediately clear that it denotes a process that does not have the DNII property. Notice that, by definition, $h \in \mathit{High}^P$ and $[l := l + 1] \in \mathit{Int}^P$. When $[l := l + 1]$ is performed, this cannot be observed in the externally observable process because $[l := l + 1]$ is an internal action. This means that, irrespective of the value that is initially assigned to h, the externally observable process performs either a or b and after that terminates successfully. This is why the process denoted by P may seem to have the DNII property. However, at the point that the externally observable process has the option to perform a, it has also the option to perform b in the case where the value initially assigned to h is not 0, while it does not have also the option to perform b in the case where the value initially assigned to h is 0. In other words, the externally observable process in the former case differs from the externally observable process in the latter case. This means that, whether or not 0 is initially assigned to h can be inferred from the externally observable process. Hence, the informal conclusion is that P denotes a process that does not have the DNII property. More formally, this conclusion follows from the definition of the DNII property: we have

$$\mathrm{ACP}_\epsilon^\tau\text{-I+REC+CFAR} \vdash \tau_{\mathit{Int}^P}(\mathsf{V}_\sigma(P)) = \tau \cdot a + b$$

for all evaluation maps σ such that $\sigma(h) = 0$ and we have

$$\mathrm{ACP}_\epsilon^\tau\text{-I+REC+CFAR} \vdash \tau_{\mathit{Int}^P}(\mathsf{V}_{\sigma'}(P)) = a + b$$

for all evaluation maps σ' such that $\sigma'(h) \neq 0$, but we do not have

$$\mathrm{ACP}_\epsilon^\tau\text{-I+REC+CFAR} \vdash \tau \cdot a + b = a + b\,.$$

In CSP_σ [16], presumably the only imperative process algebra that supports abstraction from actions that are considered not to be visible, the DNII property cannot be defined. The cause of this is that abstraction from actions that are considered not to be visible means in CSP_σ that these actions are simply removed. Because of that processes such as $\tau_{Int^P}(\mathsf{V}_\sigma(P))$ and $\tau_{Int^P}(\mathsf{V}_{\sigma'}(P))$ from the example given above are equated.

Assume that $h, l \in \mathcal{V}$ and $a, b \in \mathsf{A}$. Let Q be the closed $\mathrm{ACP}_\epsilon^\tau$-I+REC term

$$(h = 0) :\rightarrow [l := l + 1] \cdot a + \neg(h = 0) :\rightarrow [l := l - 1] \cdot a + b$$

of sort $\mathbf{P}$ with $Low^Q = \{l\}$ and $Ext^Q = \{a, b\}$. Q is a very simple example of a term that denotes a process that has the DNII property. This follows from the definition of the DNII property: we have

$$\mathrm{ACP}_\epsilon^\tau\text{-I+REC+CFAR} \vdash \tau_{Int^Q}(\mathsf{V}_\sigma(Q)) = \tau \cdot a + b$$

for all evaluation maps σ such that $\sigma(h) = 0$, we have

$$\mathrm{ACP}_\epsilon^\tau\text{-I+REC+CFAR} \vdash \tau_{Int^Q}(\mathsf{V}_{\sigma'}(Q)) = \tau \cdot a + b$$

for all evaluation maps σ' such that $\sigma'(h) \neq 0$, and we trivially have

$$\mathrm{ACP}_\epsilon^\tau\text{-I+REC+CFAR} \vdash \tau \cdot a + b = \tau \cdot a + b\,.$$

The DNII property is only one of the process properties related to information flow security that can be defined and verified in $\mathrm{ACP}_\epsilon^\tau$-I+REC+CFAR. Insofar as information flow security of contemporary systems is concerned, it 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, 36, 11, 31]). The problem with those process algebras is that state components must be mimicked by processes in them.

The DNII property concerns the non-disclosure of confidential data, contained in state components of a system, through the possibly ongoing interaction of the system concerned with its environment. To my knowledge, such a property has not been proposed in the literature on information-flow security before. However, the DNII property is reminiscent of the combination of data non-interference and event non-interference as defined in [37], 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. This process algebra distinguishes itself from imperative process algebras such as VPLA [27], IPAL [33], CSP$_\sigma$ [16], AWN [19], and the process algebra proposed in [13] by the following three properties: (1) it supports abstraction from actions that are considered not to be visible; (2) 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; (3) it offers the possibility of equational verification of process equivalence.

Properties (1)–(3) have been achieved by the inclusion of the silent step constant τ and the abstraction operators τ_I in the constants and operators of the process algebra, the use of the rooted branching bisimulation equivalence relation $\leftrightarroweq_{\mathsf{rb}}$ for the equivalence of processes in its semantics, and the provision of the equational axiomatization of $\leftrightarroweq_{\mathsf{rb}}$.

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 [7] and the data parameterized action operators have been inspired by [8].

Acknowledgement

I thank two anonymous referees for carefully reading a preliminary version of this paper, for suggesting improvements of the presentation of the paper, and for pointing out two minor but annoying errors in it.

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. This appendix is strongly based on Section 4 of [9].

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 stands for an arbitrary subset of $\mathcal{A}$ or the set $\mathcal{A}_\tau$, I stands for an arbitrary subset 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.

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 $t\ {}^{\{\phi\}}\!\downarrow$.

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

$$\frac{}{\alpha \xrightarrow{\{\mathsf{t}\}\,\alpha} \epsilon} \qquad \frac{}{\epsilon \ \{\mathsf{t}\}\!\downarrow} \qquad \frac{x \ \{\phi\}\!\downarrow}{x + y \ \{\phi\}\!\downarrow} \qquad \frac{y \ \{\phi\}\!\downarrow}{x + y \ \{\phi\}\!\downarrow} \qquad \frac{x \xrightarrow{\{\phi\}\,\alpha} x'}{x + y \xrightarrow{\{\phi\}\,\alpha} x'} \qquad \frac{y \xrightarrow{\{\phi\}\,\alpha} y'}{x + y \xrightarrow{\{\phi\}\,\alpha} y'}$$

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

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

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

$$\frac{x \xrightarrow{\{\phi\}\,a(e_1,\ldots,e_n)} x',\ y \xrightarrow{\{\psi\}\,b(e'_1,\ldots,e'_n)} y'}{x \parallel y \xrightarrow{\{\phi\wedge\psi\wedge e_1=e'_1\wedge\ldots\wedge e_n=e'_n\}\,c(e_1,\ldots,e_n)} x' \parallel y'}\ \gamma(a,b) = c,\ \mathfrak{D} \not\models \phi \wedge \psi \wedge e_1 = e'_1 \wedge \ldots \wedge e_n = e'_n \Leftrightarrow \mathsf{f}$$

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

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

$$\frac{x \xrightarrow{\{\phi\}\,a(e_1,\ldots,e_n)} x',\ y \xrightarrow{\{\psi\}\,b(e'_1,\ldots,e'_n)} y'}{x \mid y \xrightarrow{\{\phi\wedge\psi\wedge e_1=e'_1\wedge\ldots\wedge e_n=e'_n\}\,c(e_1,\ldots,e_n)} x' \parallel y'}\ \gamma(a,b) = c,\ \mathfrak{D} \not\models \phi \wedge \psi \wedge e_1 = e'_1 \wedge \ldots \wedge e_n = e'_n \Leftrightarrow \mathsf{f}$$

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

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

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

$$\frac{x \ \{\phi\}\!\downarrow}{\mathsf{V}_\sigma(x) \ \{\sigma(\phi)\}\!\downarrow}\ \mathfrak{D} \not\models \sigma(\phi) \Leftrightarrow \mathsf{f} \qquad \frac{x \xrightarrow{\{\phi\}\,\tau} x'}{\mathsf{V}_\sigma(x) \xrightarrow{\{\sigma(\phi)\}\,\tau} \mathsf{V}_\sigma(x')}\ \mathfrak{D} \not\models \sigma(\phi) \Leftrightarrow \mathsf{f}$$

$$\frac{x \xrightarrow{\{\phi\}\,a} x'}{\mathsf{V}_\sigma(x) \xrightarrow{\{\sigma(\phi)\}\,a} \mathsf{V}_\sigma(x')}\ \mathfrak{D} \not\models \sigma(\phi) \Leftrightarrow \mathsf{f} \qquad \frac{x \xrightarrow{\{\phi\}\,a(e_1,\ldots,e_n)} x'}{\mathsf{V}_\sigma(x) \xrightarrow{\{\sigma(\phi)\}\,a(\sigma(e_1),\ldots,\sigma(e_n))} \mathsf{V}_\sigma(x')}\ \mathfrak{D} \not\models \sigma(\phi) \Leftrightarrow \mathsf{f}$$

$$\frac{x \xrightarrow{\{\phi\}\,[v:=e]} x'}{\mathsf{V}_\sigma(x) \xrightarrow{\{\sigma(\phi)\}\,[v:=\sigma(e)]} \mathsf{V}_{\sigma\{\sigma(e)/v\}}(x')}\ \mathfrak{D} \not\models \sigma(\phi) \Leftrightarrow \mathsf{f}$$

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

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 $\phi \vee \psi :\to a$ and $\phi :\to a + \psi :\to a$: 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 observable action a is performed before anything has been observed, whereas this potential is a 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 [7] and the second issue alone can be dealt with by means of the notion of branching bisimulation equivalence introduced in [41] 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 $\mathrm{ACP}_\epsilon^\tau$-I terms of sort $\mathbf{C}$, is used for the $\mathrm{ACP}_\epsilon^\tau$-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 $t_1 \stackrel{\{\phi\}}{\downarrow}$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \Rightarrow \bigvee \Psi$ and, for all $\psi \in \Psi$, 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 $t_2^n \stackrel{\{\psi'\}}{\downarrow}$;

- if $t_2 \stackrel{\{\phi\}}{\downarrow}$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \Rightarrow \bigvee \Psi$ and, for all $\psi \in \Psi$, 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 $t_1^n \stackrel{\{\psi'\}}{\downarrow}$.

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 $\mathfrak{D} \models \phi \Rightarrow \bigvee \Psi$ 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 $\mathfrak{D} \models \phi \Rightarrow \bigvee \Psi$ 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 $t_1 \stackrel{\{\phi\}}{\downarrow}$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \Rightarrow \bigvee \Psi$ and, for all $\psi \in \Psi$, $t_2 \stackrel{\{\psi\}}{\downarrow}$;

- if $t_2 \stackrel{\{\phi\}}{\downarrow}$, then there exists a finite set $\Psi \subseteq \mathcal{C}^{sat}$ such that $\mathfrak{D} \models \phi \Rightarrow \bigvee \Psi$ and, for all $\psi \in \Psi$, $t_1 \stackrel{\{\psi\}}{\downarrow}$.

Two terms $t_1, t_2 \in \mathcal{P}_{rec}$ are *rooted ab-bisimulation equivalent*, written $t_1 \leftrightarrows_{\mathsf{rab}} t_2$, 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 [7]. In the absence of all terms of sort $\mathbf{C}$ other than the constants t and f, rooted

ab-bisimulation equivalence is essentially the same as rooted branching bisimulation equivalence as defined in [41].

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

References

[1] Jos C. M. Baeten and Jan A. Bergstra. Global renaming operators in concrete process algebra. *Information and Computation*, 78(3):205–245, 1988. doi:10.1016/0890-5401(88)90027-2.

[2] Jos C. M. Baeten and Jan A. Bergstra. Process algebra with signals and conditions. In Manfred Broy, editor, *Programming and Mathematical Method*, pages 273–323, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. doi:10.1007/978-3-642-77572-7_13.

[3] Jos C. M. Baeten and W. P. Weijland. *Process algebra*, volume 18 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1990. doi:10.1017/CBO9780511624193.

[4] Twan Basten. Branching bisimilarity is an equivalence indeed! *Information Processing Letters*, 58(3):141–147, 1996. doi:10.1016/0020-0190(96)00034-8.

[5] D. E. Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corporation, 1973.

[6] Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984. doi:10.1016/S0019-9958(84)80025-X.

[7] Jan A. Bergstra and Cornelis A. Middelburg. Splitting bisimulations and retrospective conditions. *Information and Computation*, 204(7):1083–1138, 2006. doi:10.1016/j.ic.2006.03.003.

[8] Jan A. Bergstra and Cornelis A. Middelburg. A process calculus with finitary comprehended terms. *Theory of Computing Systems*, 53(4):645–668, 2013. doi:10.1007/s00224-013-9468-x.

[9] Jan A. Bergstra and Cornelis A. Middelburg. Using Hoare logic in a process algebra setting. *Fundamenta Informaticae*, 179(4):321–344, 2021. doi:10.3233/FI-2021-2026.

[10] Aaron Bohannon, Benjamin C. Pierce, Vilhelm Sjöberg, Stephanie Weirich, and Steve Zdancewic. Reactive noninterference. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009*, pages 79–90. ACM, 2009. doi:10.1145/1653662.1653673.

[11] Annalisa Bossi, Riccardo Focardi, Carla Piazza, and Sabina Rossi. Verifying persistent security properties. *Computer Languages, Systems & Structures*, 30(3-4):231–258, 2004. doi:10.1016/j.cl.2004.02.005.

[12] Gérard Boudol and Ilaria Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1-2):109–130, 2002. doi:10.1016/S0304-3975(02)00010-5.

[13] Mark Bouwman, Bas Luttik, Wouter Schols, and Tim A. C. Willemse. A process algebra with global variables. In Ornela Dardha and Jurriaan Rot, editors, *Proceedings Combined 27th International Workshop on Expressiveness in Concurrency and 17th Workshop on Structural Operational Semantics, EXPRESS/SOS 2020*, volume 322 of *Electronic Proceedings in Theoretical Computer Science*, pages 33–50, 2020. doi:10.4204/EPTCS.322.5.

[14] Ellis S. Cohen. Information transmission in computational systems. In Saul Rosen and Peter J. Denning, editors, *Proceedings of the Sixth Symposium on Operating System Principles, SOSP 1977*, pages 133–139. ACM, 1977. doi:10.1145/800214.806556.

[15] Ellis S Cohen. Information transmission in sequential programs. In Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, editors, *Foundations of secure computation*, pages 297–335. Academic Press, 1978.

[16] Robert Colvin and Ian J. Hayes. CSP with hierarchical state. In Michael Leuschel and Heike Wehrheim, editors, *7th International Conference on Integrated Formal Methods, IFM 2009,*, volume 5423 of *Lecture Notes in Computer Science*, pages 118–135. Springer, 2009. doi:10.1007/978-3-642-00255-7_9.

[17] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976. doi:10.1145/360051.360056.

[18] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977. doi:10.1145/359636.359712.

[19] Ansgar Fehnker, Rob J. van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann, and Wee Lum Tan. A process algebra for wireless mesh networks. In Helmut Seidl, editor, *21st European Symposium on Programming on Programming Languages and Systems, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012*, volume 7211 of *Lecture Notes in Computer Science*, pages 295–315. Springer, 2012. doi:10.1007/978-3-642-28869-2_15.

[20] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995. doi:10.3233/JCS-1994/1995-3103.

[21] Riccardo Focardi, Sabina Rossi, and Andrei Sabelfeld. Bridging language-based and process calculi security. In Vladimiro Sassone, editor, *8th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005*, volume 3441 of *Lecture Notes in Computer Science*, pages 299–315. Springer, 2005. doi:10.1007/978-3-540-31982-5_19.

[22] Wan J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000. doi:10.1007/978-3-662-04293-9.

[23] Wan J. Fokkink. Rooted branching bisimulation as a congruence. *Journal of Computer and System Sciences*, 60(1):13–37, 2000. doi:10.1006/jcss.1999.1663.

[24] Joseph A. Goguen. Theorem proving and algebra. *CoRR*, abs/2101.02690, 2021. arXiv:2101.02690.

[25] Jan Friso Groote and Alban Ponse. Process algebra with guards: Combining hoare logic with process algebra. *Formal Aspects of Computing*, 6(2):115–164, 1994. doi:10.1007/BF01221097.

[26] David Harel and Amir Pnueli. On the development of reactive systems. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems*,

volume F13 of *NATO ASI Series*, pages 477–498. Springer, 1984. doi:10.1007/978-3-642-82453-1_17.

[27] Matthew Hennessy and Anna Ingólfsdóttir. Communicating processes with value-passing and assignments. *Formal Aspects of Computing*, 5(5):432–466, 1993. doi:10.1007/BF01212486.

[28] Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995. doi:10.1016/0304-3975(94)00172-F.

[29] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. *ACM Transactions on Programming Languages and Systems*, 29(6):31, 2007. doi:10.1145/1286821.1286822.

[30] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994. doi:10.1145/177492.177726.

[31] Gavin Lowe. Semantic models for information flow. *Theoretical Computer Science*, 315(1):209–256, 2004. doi:10.1016/j.tcs.2003.11.019.

[32] Mohammad Reza Mousavi, Michel A. Reniers, and Jan Friso Groote. Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1):107–147, 2005. doi:10.1016/j.ic.2005.03.002.

[33] Rocco De Nicola and Rosario Pugliese. Testing semantics of asynchronous distributed programs. In Mads Dam, editor, *5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages, Selected Papers*, volume 1192 of *Lecture Notes in Computer Science*, pages 320–344. Springer, 1996. doi:10.1007/3-540-62503-8_15.

[34] Kevin R. O'Neill, Michael R. Clarkson, and Stephen Chong. Information-flow security for interactive programs. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006)*, pages 190–201. IEEE Computer Society, 2006. doi:10.1109/CSFW.2006.16.

[35] Don Pigozzi and Antonino Salibra. The abstract variable-binding calculus. *Studia Logica*, 55(1):129–179, 1995. doi:10.1007/BF01053036.

[36] Peter Y. A. Ryan and Steve A. Schneider. Process algebra and non-interference. *Journal of Computer Security*, 9(1/2):75–103, 2001. doi:10.3233/JCS-2001-91-204.

[37] Najah Ben Said and Ioana Cristescu. End-to-end information flow security for web services orchestration. *Science of Computer Programming*, 187:102376, 2020. doi:10.1016/j.scico.2019.102376.

[38] Donald Sannella and Andrzej Tarlecki. Algebraic preliminaries. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, IFIP State-of-the-Art Reports, pages 13–30. Springer, 1999. doi:10.1007/978-3-642-59851-7_2.

[39] Fred B. Schneider. *On Concurrent Programming*. Graduate Texts in Computer Science. Springer, 1997. doi:10.1007/978-1-4612-1830-2.

[40] Geoffrey Smith and Dennis M. Volpano. Secure information flow in a multi-threaded imperative language. In David B. MacQueen and Luca Cardelli, editors, *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '98*, pages 355–364. ACM, 1998. doi:10.1145/268946.268975.

[41] Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996. doi:10.1145/233551.233556.

[42] Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996. doi:10.3233/JCS-1996-42-304.

[43] Martin Wirsing. Algebraic specification. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 675–788. Elsevier and MIT Press, 1990. doi:10.1016/b978-0-444-88074-1.50018-4.