A Concurrent Calculus with Atomic Transactions

Lucia Acciai[1], Michele Boreale[2], and Silvano Dal Zilio[1]

[1] Laboratoire d'Informatique Fondamentale de Marseille (LIF),
CNRS and Université de Provence, France
[2] Dipartimento di Sistemi e Informatica, Università di Firenze, Italy

Abstract. The *Software Transactional Memory* (STM) model is an original approach for controlling concurrent accesses to ressources without the need for explicit lock-based synchronization mechanisms. A key feature of STM is to provide a way to group sequences of read and write actions inside *atomic blocks*, similar to database transactions, whose whole effect should occur atomically.
In this paper, we investigate STM from a process algebra perspective and define an extension of asynchronous CCS with atomic blocks of actions. Our goal is not only to set a formal ground for reasoning on STM implementations but also to understand how this model fits with other concurrency control mechanisms. We also view this calculus as a test bed for extending process calculi with atomic transactions. This is an interesting direction for investigation since, for the most part, actual works that mix transactions with process calculi consider compensating transactions, a model that lacks all the well-known ACID properties.
We show that the addition of atomic transactions results in a very expressive calculus, enough to easily encode other concurrent primitives such as guarded choice and multiset-synchronization (à la join-calculus). The correctness of our encodings is proved using a suitable notion of bisimulation equivalence. The equivalence is then applied to prove interesting "laws of transactions" and to obtain a simple normal form for transactions.

1 Introduction

The craft of programming concurrent applications is about mastering the strains between two key factors: getting hold of results as quickly as possible, while ensuring that only correct results (and behaviors) are observed. To this end, it is vital to avoid unwarranted access to shared resources. The *Software Transactional Memory* (STM) model is an original approach for controlling concurrent accesses to resources without using explicit lock-based synchronization mechanisms. Similarly to database transactions, the STM approach provides a way to group sequences of read and write actions inside *atomic blocks* whose whole effect should occur atomically. The STM model has several advantages. Most notably, it dispenses the programmer with the need to explicitly manipulate locks, a task widely recognized as difficult and error-prone. Moreover, atomic transactions provide a clean conceptual basis for concurrency control, which should ease the verification of concurrent programs. Finally, the model is effective: there exist several STM implementations for designing software for multiprocessor systems; these applications exhibit good performances in practice (compared to equivalent, hand-crafted, code using locks).

We investigate the STM model from a process algebra perspective and define an extension of asynchronous CCS [22] with atomic blocks of actions. We call this calculus AtCCS. The choice of a dialect of CCS is motivated by an attention to economy: to focus on STM primitives, we study a calculus as simple as possible and dispense with orthogonal issues such as values,

mobility of names or processes, *etc*. We believe that our work could be easily transferred to a richer setting. Our goal is not only to set a formal ground for reasoning on STM implementations but also to understand how this model fits with other concurrency control mechanisms. We also view this calculus as a test bed for extending process calculi with atomic transactions. This is an interesting direction for investigation since, for the most part, works that mix transactions with process calculi consider *compensating transactions*, see e.g. [2,4,6,8,9,11,12,13,21].

The idea of providing hardware support for software transactions originated from works by Herlihy and Moss [20] and was later extended by Shavit and Touitou [25] to software-only transactional memory. Transactions are used to protect the execution of an atomic block. Intuitively, each thread that enters a transaction takes a snapshot of the shared memory (the global state). The evaluation is optimistic and all actions are performed on a copy of the memory (the local state). When the transaction ends, the snapshot is compared with the current state of the memory. There are two possible outcomes: if the check indicates that concurrent writes have occurred, the transaction aborts and is rescheduled; otherwise, the transaction is committed and its effects are propagated instantaneously. Very recently, Harris et al. [19] have proposed a (combinator style) language of transactions that enables arbitrary atomic operations to be composed into larger *atomic expressions*. We base the syntax of AtCCS on the operators defined in [19].

The main contributions of this work are: (1) the definition of a process calculus with atomic transactions; and (2) the definition of an asynchronous bisimulation equivalence $\approx_a$ that allows compositional reasoning on transactions. We also have a number of more specific technical results. We show that AtCCS is expressive enough to easily encode interesting concurrent primitives, such as (preemptive versions of) guarded choice and multiset-synchronization, and the leader election problem (Section 3). Next, we define an equivalence between atomic expressions $\backsimeq$ and prove that $\approx_a$ and $\backsimeq$ are congruences (Section 4). These equivalences are used to prove the correctness of our encodings, to prove interesting "behavioral laws of transactions" and to define a simple normal form for transactions. We also show that transactions (modulo $\backsimeq$) have an algebraic structure close to that of a bound semilattice, an observation that could help improve the design of the transaction language. Finally, we propose a may-testing equivalence for AtCCS, give an equivalent characterization using a trace-based semantics and show that may testing equivalence is unable to notice the presence of transactions (Section 5). Section 6 concludes with an overview on future and related works. The proofs of the main results are reported in the appendices.

2 The calculus

We define the syntax and operational semantics of AtCCS, which is essentially a cut down version of asynchronous CCS, without choice and relabeling operators, equipped with atomic blocks and constructs for composing (transactional) sequences of actions.

Syntax of Processes and Atomic Expressions. The syntax of AtCCS, given in Table 1, is divided into syntactical categories that define a stratification of terms. The definition of the calculus depends on a set of names, ranged over by $a, b, \ldots$ As in CCS, names model communication channels used in process synchronization, but they also occur as objects of read and write actions in atomic transactions.

Atomic expressions, ranged over by $M, N, \ldots$, are used to define sequences of actions whose effect should happen atomically. Actions $\mathtt{rd}\,a$ and $\mathtt{wt}\,a$ represent attempts to input and output to the channel a. Instead of using snapshots of the state for managing transaction,

Actions	α, β	$::=$	$\texttt{rd}\, a$ (tentative) read access to a
		$\mid$	$\texttt{wt}\, a$ (tentative) write access to a
(Atomic) Expressions	M, N	$::=$	$\texttt{end}$ termination
		$\mid$	$\texttt{retry}$ abort and retry the current atomic block
		$\mid$	$\alpha.M$ action prefix
		$\mid$	$M\ \texttt{orElse}\ N$ alternative
Ongoing expressions	A, B	$::=$	$(M)_{\sigma;\delta}$ execution of M with state σ and log δ
		$\mid$	$A\ \texttt{orElse}\ B$ ongoing alternative
Processes	P, Q	$::=$	$\mathbf{0}$ nil
		$\mid$	$\overline{a}$ (asynchronous) output
		$\mid$	$a.P$ input
		$\mid$	$*\, a\, .P$ replicated input
		$\mid$	$P \mid Q$ parallel composition
		$\mid$	$P \backslash^n a$ hiding
		$\mid$	$\texttt{atom}(M)$ atomic block
		$\mid$	$\{\!\lvert A \rvert\!\}_M$ ongoing atomic block

Table 1. Syntax of AtCCS: Processes and Atomic Expressions.

we use a log-based approach. During the evaluation of an atomic block, actions are recorded in a private log δ (a sequence $\alpha_1 \ldots \alpha_n$) and have no effects outside the scope of the transaction until it is committed. The action `retry` aborts an atomic expression unconditionally and starts its execution afresh, with an empty log ε. The termination action `end` signals that an expression is finished and should be committed. If the transaction can be committed, all actions in the log are performed at the same time and the transaction is closed, otherwise the transaction aborts. Finally, transactions can be composed using the operator `orElse`, which implements (preemptive) alternatives between expressions. M `orElse` N behaves as expression N if M aborts and has the behavior of M otherwise.

Processes, ranged over by $P, Q, R, \ldots$, model concurrent systems of communicating agents. We have the usual operators of CCS: the empty process, $\mathbf{0}$, the parallel composition $P \mid Q$, and the input prefix $a.P$. There are some differences though. The calculus is asynchronous, meaning that a process cannot block on output actions. Also, we use *replicated input* $*\, a\, .P$ instead of recursion (this does not change the expressiveness of the calculus) and we lack the choice and relabeling operators of CCS. Finally, the main addition is the presence of the operator $\texttt{atom}(M)$, which models a transaction that safeguards the expression M. The process $\{\!\lvert A \rvert\!\}_M$ represents the ongoing evaluation of an atomic block M: the subscript is used to keep the initial code of the transaction, in case it is aborted and executed afresh, while A holds the remaining actions that should be performed.

An *ongoing atomic block*, $A, B, \ldots$, is essentially an atomic expression enriched with an *evaluation state* σ and a *log* δ of the currently recorded actions. A state σ is a multiset of names that represents the output actions visible to the transaction when it was initiated. (This notion

of state bears some resemblance with tuples space in coordination calculi, such as Linda [10].) When a transaction ends, the state σ recorded in the block $(M)_{\sigma;\delta}$ (the state at the initiation of the transaction) can be compared with the current state (the state when the transaction ends) to check if other processes have concurrently made changes to the global state, in which case the transaction should be aborted.

Notation. In the following, we write $\sigma \uplus \{a\}$ for the multiset σ enriched with the name a and $\sigma \setminus \sigma'$ for the multiset obtained from σ by removing elements found in σ', that is the smallest multiset σ'' such that $\sigma \subseteq \sigma' \uplus \sigma''$. The symbol $\emptyset$ stands for the empty multiset while $\{a^n\}$ is the multiset composed of exactly n copies of a, where $\{a^0\} = \emptyset$.

Given a log δ, we use the notation $\mathrm{wt}(\delta)$ for the multiset of names which appear as objects of a write action in δ. Similarly, we use the notation $\mathrm{rd}(\delta)$ for the multiset of names that are objects of read actions. The functions wt and rd may be inductively defined as follows: $\mathrm{wt}(\mathtt{wt}\, a.\delta) = \mathrm{wt}(\delta) \uplus \{a\}$; $\mathrm{rd}(\mathtt{rd}\, a.\delta) = \mathrm{rd}(\delta) \uplus \{a\}$; $\mathrm{wt}(\mathtt{rd}\, a.\delta) = \mathrm{rd}(\mathtt{wt}\, a.\delta) = \mathrm{wt}(\delta)$; and $\mathrm{wt}(\varepsilon) = \mathrm{rd}(\varepsilon) = \varepsilon$.

Example: Composing Synchronization. Before we describe the meaning of processes, we try to convey the semantics of AtCCS (and the usefulness of the atomic block operator) using a simple example. We take the example of a concurrent system with two memory cells, M_1 and M_2, used to store integers. We consider here a straightforward extension of the calculus with "value-passing[1]." In this setting, we can model a cell with value v by an output $\overline{m_i}!v$ and model an update by a process of the form $m_i?x.(\overline{m_i}!v' \mid \ldots)$. With this encoding, the channel name m_i acts as a lock protecting the shared resource M_i.

Assume now that the values of the cells should be synchronized to preserve a global invariant on the system. For instance, we model a flying aircraft, each cell store the pitch of an aileron and we need to ensure that the aileron stay aligned (that the values of the cells are equal). A process testing the validity of the invariant is for example P_1 below (we suppose that a message on the reserved channel *err* triggers an alarm). There are multiple design choices for resetting the value of both cells to 0, e.g. P_2 and P_3.

$$P_1 \triangleq m_1?x.m_2?y.\text{if } x \mathrel{!}= y \text{ then } \overline{err}!$$

$$P_2 \triangleq m_2?x.m_1?y.\big(\overline{m_1}!0 \mid \overline{m_2}!0\big) \qquad P_3 \triangleq m_1?x.\big(\overline{m_1}!0 \mid m_2?y.\overline{m_2}!0\big)$$

Each choice exemplify a problem with lock-based programming. The composition of P_1 with P_2 leads to a race condition where P_1 acquire the lock on M_1, P_2 on M_2 and each process gets stuck. The composition of P_1 and P_3 may break the invariant (the value of M_1 is updated too quickly). A solution in the first case is to strengthen the invariant and enforce an order for acquiring locks, but this solution is not viable in general and opens the door to *priority inversion* problems. Another solution is to use an additional (master) lock to protect both cells, but this approach obfuscate the code and significantly decreases the concurrency of the system.

Overall, this simple example shows that synchronization constraints do not compose well when using locks. This situation is consistently observed (and bears a resemblance to the inheritance anomaly problem found in concurrent object-oriented languages). The approach advocated in this paper is to use atomic transactions. In our example, the problem is solved by simply wrapping the two operations in a transaction, like in the following process, which en-

[1] Keeping to our attention to economy in the definition of AtCCS, we choose not to consider values in the formal syntax, but our results could be easily extended to take them into account.

$$\text{(out)}\ \overline{a};\sigma \to \mathbf{0};\sigma \uplus \{a\} \qquad \text{(rep)}\ *a.P;\sigma \uplus \{a\} \to P \mid *a.P;\sigma$$

$$\text{(in)}\ a.P;\sigma \uplus \{a\} \to P;\sigma \qquad \text{(com)}\ \frac{P;\sigma \to P';\sigma \uplus \{a\} \quad Q;\sigma \uplus \{a\} \to Q';\sigma}{P \mid Q \to P' \mid Q'}$$

$$\text{(parL)}\ \frac{P;\sigma \to P';\sigma'}{P \mid Q;\sigma \to P' \mid Q;\sigma'} \qquad \text{(hid)}\ \frac{P;\sigma \uplus \{a^n\} \to P';\sigma' \uplus \{a^m\} \qquad a \notin \sigma,\sigma'}{P \setminus^n a;\sigma \to P' \setminus^m a;\sigma'}$$

$$\text{(parR)}\ \frac{Q;\sigma \to Q';\sigma'}{P \mid Q;\sigma \to P \mid Q';\sigma'} \qquad \text{(atSt)}\ \mathtt{atom}(M);\sigma \to \{\!|(M)_{\sigma;\varepsilon}|\!\}_M;\sigma$$

$$\text{(atPass)}\ \frac{A \to A'}{\{\!|A|\!\}_M;\sigma \to \{\!|A'|\!\}_M;\sigma} \qquad \text{(atRe)}\ \{\!|(\mathtt{retry})_{\sigma';\delta}|\!\}_M;\sigma \to \mathtt{atom}(M);\sigma$$

$$\text{(atFail)}\ \frac{\mathrm{rd}(\delta) \not\subseteq \sigma}{\{\!|(\mathtt{end})_{\sigma';\delta}|\!\}_M;\sigma \to \mathtt{atom}(M);\sigma}$$

$$\text{(atOk)}\ \frac{\mathrm{rd}(\delta) \subseteq \sigma \quad \sigma = \sigma'' \uplus \mathrm{rd}(\delta) \quad \mathrm{wt}(\delta) = \{a_1,\ldots,a_n\}}{\{\!|(\mathtt{end})_{\sigma';\delta}|\!\}_M;\sigma \to \overline{a_1} \mid \cdots \mid \overline{a_n};\sigma''}$$

Table 2. Operational Semantics: Processes.

sures that all cell updates are effected atomically.

$$\mathtt{atom}\big(\mathtt{rd}\,(m_2?y).\mathtt{wt}\,(m_2!0).\mathtt{rd}\,(m_1?x).\mathtt{wt}\,(m_1!0)\big)$$

More examples may be found on the paper on composable memory transactions [19], which makes a compelling case that "even correctly-implemented concurrency abstractions cannot be composed together to form larger abstractions."

Operational Semantics. Like for the syntax, the semantics of AtCCS is stratified in two levels: there is one reduction relation for processes and a second for atomic expressions. With a slight abuse of notation, we use the same symbol ($\to$) for both relations.

Reduction for Processes. Table 2 gives the semantics of processes. A reduction is of the form $P;\sigma \to P';\sigma'$ where σ is the state of P. The state σ records the names of all output actions visible to P when reduction happens. It grows when an output is reduced, (out), and shrinks in the case of inputs, (in) and (rep). A parallel composition evolves if one of the component evolves or if both can synchronize, rules (parL), (parR) and (com). In a hiding $P \setminus^n a$, the annotation n is an integer denoting the number of outputs on a which are visible to P. Intuitively, in a "configuration" $P \setminus^n a;\sigma$, the outputs visible to P are those in $\sigma \uplus \{a^n\}$. This extra annotation is necessary because the scope of a is restricted to P, hence it is not possible to have outputs on a in the global state. Rule (hid) allows synchronization on the name a to happen inside a hiding. For instance, we have $(P \mid \overline{a}) \setminus^n a;\sigma \to P \setminus^{n+1} a;\sigma$.

The remaining reduction rules govern the evolution of atomic transactions. Like in the case of (com), all those rules, but (atOk), leave the global state unchanged. Rule (atSt) deals with the initiation of an atomic block $\mathtt{atom}(M)$: an ongoing block $\{\!|(M)_{\sigma;\varepsilon}|\!\}_M$ is created which holds the current evaluation state σ and an empty log ε. An atomic block $\{\!|A|\!\}_M$ reduces when its expression A reduces, rule (atPass). (The reduction relation for ongoing expression is defined by the rules in Table 3.) Rules (atRe), (atFail) and (atOk) deal with the completion of a transaction. After a finite number of transitions, the evaluation of an ongoing expression will necessarily

$$\text{(ARdOk)}\ \frac{\mathrm{rd}(\delta) \uplus \{a\} \subseteq \sigma}{(\mathtt{rd}\, a\, .M)_{\sigma;\delta} \to (M)_{\sigma;\delta.\mathtt{rd}\, a}} \qquad \text{(ARdF)}\ \frac{\mathrm{rd}(\delta) \uplus \{a\} \not\subseteq \sigma}{(\mathtt{rd}\, a\, .M)_{\sigma;\delta} \to (\mathtt{retry})_{\sigma;\delta}}$$

$$\text{(AWr)}\quad (\mathtt{wt}\, a\, .M)_{\sigma;\delta} \to (M)_{\sigma;\delta.\mathtt{wt}\, a}$$

$$\text{(AOI)}\quad (M_1\ \mathtt{orElse}\ M_2)_{\sigma;\delta} \to (M_1)_{\sigma;\delta}\ \mathtt{orElse}\ (M_2)_{\sigma;\delta}$$

$$\text{(AOF)}\ (\mathtt{retry})_{\sigma;\delta}\ \mathtt{orElse}\ B \to B \qquad \text{(AOE)}\ (\mathtt{end})_{\sigma;\delta}\ \mathtt{orElse}\ B \to (\mathtt{end})_{\sigma;\delta}$$

$$\text{(AOL)}\ \frac{A \to A'}{A\ \mathtt{orElse}\ B \to A'\ \mathtt{orElse}\ B} \qquad \text{(AOR)}\ \frac{B \to B'}{A\ \mathtt{orElse}\ B \to A\ \mathtt{orElse}\ B'}$$

Table 3. Operational Semantics : Ongoing Atomic Expression.

result in a fail state, $(\mathtt{retry})_{\sigma;\delta}$, or a success, $(\mathtt{end})_{\sigma;\delta}$. In the first case, rule (atRe), the transaction is aborted and started again from scratch. In the second case, we need to check if the log is consistent with the current evaluation state. A log is consistent if the read actions of δ can be performed on the current state. If the check fails, rule (atFail), the transaction aborts. Otherwise, rule (atOk), we commit the transaction: the names in $\mathrm{rd}(\delta)$ are taken from the current state and a bunch of outputs on the names in $\mathrm{wt}(\delta)$ are generated.

Reduction for Ongoing Expressions. Table 3 gives the semantics of ongoing atomic expressions. We recall that, in an expression $(\mathtt{rd}\, a\, .M)_{\sigma;\delta}$, the subscript σ is the *initial state*, that is a copy of the state at the time the block has been created and δ is the log of actions performed since the initiation of the transaction.

Rule (ARdOk) states that a read action $\mathtt{rd}\, a$ is recorded in the log δ if all the read actions in $\delta.\mathtt{rd}\, a$ can be performed in the initial state. If it is not the case, the ongoing expression fails, rule (ARdF). This test may be interpreted as a kind of optimization: if a transaction cannot commit in the initial state then, should it commit at the end of the atomic block, it would mean that the global state has been concurrently modified during the execution of the transaction. Note that we consider the initial state σ and not $\sigma \uplus \mathrm{wt}(\delta)$, which means that, in an atomic block, write actions are not directly visible (they cannot be consumed by a read action). This is coherent with the fact that outputs on $\mathrm{wt}(\delta)$ only take place after commit of the block. Rule (AWr) states that a write action always succeeds and is recorded in the current log.

The remaining rules govern the semantics of the `retry`, `end` and `orElse` constructs. These constructs are borrowed from the STM combinators used in the implementation of an STM system in Concurrent Haskell [19]. We define these operators with an equivalent semantics, with the difference that, in our case, a state is not a snapshot of the (shared) memory but a multiset of visible outputs. A composition $M\ \mathtt{orElse}\ N$ corresponds to the interleaving of the behaviors of M and N, which are independently evaluated with respect to the same evaluation state (but have distinct logs). The `orElse` operator is preemptive: the ongoing block $M\ \mathtt{orElse}\ N$ ends if and only M ends or M aborts and N ends.

3 Encoding Concurrency Primitives

Our first example is a simple solution to the celebrated *leader election* problem that does not yield to deadlock and ensures that, at each round, a leader is elected.

Consider a system composed by n processes and a token, named t, that is modeled by an output $\bar{t}$. A process becomes a leader by getting (making an input on) t. As usual, all participants run the same process (except for the value of their identity). We suppose that there is only one copy of the token in the system and that leadership of process i is communicated to the other processes by outputting on a reserved name win_i. A participant that is not a leader output on $loose_i$. The protocol followed by the participants is defined by the following process:

$$L_i \triangleq \big(\texttt{atom}\big(\texttt{rd}\,t\,.\texttt{wt}\,k\,.\texttt{end}\ \texttt{orElse}\ \texttt{wt}\,k'\,.\texttt{end}\big) \mid k.(\overline{win_i} \mid \bar{t}) \mid k'.\overline{loose_i}\big) \setminus^0 k \setminus^0 k'$$

In this encoding, the atomic block is used to protect the concurrent accesses to t. If the process L_i commits its transaction and inputs (grabs) the token, it immediately release an output on its private channel k. The transactions of the other participants may either fail or commit while releasing an output on their private channel k'. Then, the elected process L_i may proceed with a synchronization on k that triggers the output $\overline{win_i}$ and release the lock. The semantics of $\texttt{atom}(\,)$ ensures that only one transaction can acquire the lock and commit the atomic block, then no other process have acquired the token in the same round and we are guaranteed that there could be at most one leader.

This expressivity result is mixed blessing. Indeed, it means that any implementation of the atomic operator should be able to solve the leader election problem, which is known to be very expensive in the case of loosely-coupled systems or in presence of failures (see e.g. [24] for a discussion on the expressivity of process calculi and electoral systems). On the other hand, atomic transactions are optimistic and are compatible with the use of probabilistic approaches. Therefore it is still reasonable to expect a practical implementation of AtCCS.

In the following, we show how to encode two fundamental concurrency patterns, namely (preemptive versions of) the choice and join-pattern operators.

Guarded choice. We consider an operator for choice, $\mu_1.P_1 + \cdots + \mu_n.P_n$, such that every process is prefixed by an action μ_i that is either an output $\overline{a}_i$ or an input a_i. The semantics of choice is characterized by the following three reduction rules (we assume that Q is also a choice):

$$\text{(c-inp)}\ a.P + Q;\sigma \uplus \{a\} \to P;\sigma \qquad \text{(c-out)}\ \overline{a}.P + Q;\sigma \to P;\sigma \uplus \{a\}$$

$$\text{(c-pass)}\quad \frac{a \notin \sigma \quad Q;\sigma \to Q';\sigma'}{a.P + Q;\sigma \to Q';\sigma'}$$

A minor difference with the behavior of the choice operator found in CCS is that our semantics gives precedence to the leftmost process (this is reminiscent of the preemptive behavior of `orElse`). Another characteristic is related to the asynchronous nature of the calculus, see rule (c-out): since an output action can always interact with the environment, a choice $\overline{a}.P + Q$ may react at once and release the process $\overline{a} \mid P$.

Like in the example of the leader election problem, we can encode a choice $\mu_1.P_1 + \cdots + \mu_n.P_n$ using an atomic block that will mediate the interaction with the actions $\mu_1, \ldots, \mu_n$. We start by defining a straightforward encoding of input/output actions into atomic actions: $[\![\overline{a}]\!] = \texttt{wt}\,a$ and $[\![a]\!] = \texttt{rd}\,a$. Then the encoding of choice is the process

$$[\![\mu_1.P_1 + \cdots + \mu_n.P_n]\!] \triangleq \big(\texttt{atom}\big([\![\mu_1]\!].[\![\bar{k}_1]\!].\texttt{end}\ \texttt{orElse}\ \cdots\ \texttt{orElse}\ [\![\mu_n]\!].[\![\bar{k}_n]\!].\texttt{end}\big) \mid k_1.[\![P_1]\!] \mid \cdots \mid k_n.[\![P_n]\!]\big) \setminus^0 k_1 \ldots \setminus^0 k_n$$

The principle of the encoding is essentially the same that in our solution to the leader election problem. Actually, using the encoding for choice, we can rewrite our solution in the following form: $L_i \triangleq t.(\overline{win_i} \mid \bar{t}) + \overline{loose_i}.\mathbf{0}$. Using the rules in Table 2, it is easy to see that our encoding of choice is compatible with rule (c-inp), meaning that:

$$\begin{aligned}
[\![a.P+Q]\!];\sigma \uplus \{a\} &\rightarrow^* \left(\{\!|(\mathtt{end})_{\sigma\uplus\{a\};\mathtt{rd}\,a\,.\,\mathtt{wt}\,k_1}|\!\}_M \mid k_1.[\![P]\!] \mid \ldots\right) \setminus^0 k_1 \setminus \ldots;\sigma \uplus \{a\} \\
&\rightarrow \left(\overline{k_1} \mid k_1.[\![P]\!] \mid \ldots\right) \setminus^0 k_1 \setminus \ldots;\sigma \\
&\rightarrow \left([\![P]\!] \mid \ldots\right) \setminus^0 k_1 \setminus \ldots;\sigma
\end{aligned}$$

where the processes in parallel with $[\![P]\!]$ are harmless. In the next section, we define a weak bisimulation equivalence $\approx_a$ that can be used to garbage collect harmless processes in the sense that, e.g. $(P \mid k.Q) \setminus^0 k \approx_a P$ if P has no occurrences of k. Hence, we could prove that $[\![a.P+Q]\!];\sigma \uplus \{a\} \rightarrow^* \approx_a [\![P]\!];\sigma$, which is enough to show that our encoding is correct with respect to rule (c-inp). The same is true for rules (c-out) and (c-pass).

Join Patterns. A multi-synchronization $(a_1 \times \cdots \times a_n).P$ may be viewed as an extension of input prefix in which communication requires a synchronization with the n outputs $\overline{a_1}, \ldots, \overline{a_n}$ at once. that is, we have the reduction:

$$\text{(j-inp)} \quad (a_1 \times \cdots \times a_n).P;\sigma \uplus \{a_1, \ldots, a_n\} \rightarrow P;\sigma$$

This synchronization primitive is fundamental to the definition of the Gamma calculus of Banâtre and Le Métayer and of the Join calculus of Fournet and Gonthier. It is easy to see that the encoding of a multi-synchronization (input) is a simple transaction:

$$[\![\left(a_1 \times \cdots \times a_n\right).P]\!] \triangleq \left(\mathtt{atom}([\![a_1]\!].\cdots.[\![a_n]\!].[\![\bar{k}]\!].\mathtt{end}) \mid k.[\![P]\!]\right) \setminus^0 k \quad \text{(where } k \text{ is fresh)}$$

and that we have $[\![\left(a_1 \times \cdots \times a_n\right).P]\!];\sigma \uplus \{a_1, \ldots, a_n\} \rightarrow^* \left(\mathbf{0} \mid [\![P]\!]\right) \setminus^0 k;\sigma$, where the process $\left(\mathbf{0} \mid [\![P]\!]\right) \setminus^0 k$ is behaviorally equivalent to $[\![P]\!]$, that is:

$$[\![\left(a_1 \times \cdots \times a_n\right).P]\!];\sigma \uplus \{a_1, \ldots, a_n\} \rightarrow^* \approx_a [\![P]\!];\sigma$$

Based on this encoding, we can define two interesting derived operators: a mixed version of multi-synchronization, $(\mu_1 \times \cdots \times \mu_n).P$, that mixes input and output actions; and a replicated version, that is analogous to replicated input.

$$\begin{aligned}
[\![\left(\mu_1 \times \cdots \times \mu_n\right).P]\!] &\triangleq \left(\mathtt{atom}([\![\mu_1]\!].\cdots.[\![\mu_n]\!].[\![\bar{k}]\!].\mathtt{end}) \mid k.[\![P]\!]\right) \setminus^0 k \\
[\![*\left(\mu_1 \times \cdots \times \mu_n\right).P]\!] &\triangleq \left(\bar{r} \mid *r.\mathtt{atom}([\![\mu_1]\!].\cdots.[\![\mu_n]\!].[\![\bar{r}]\!].[\![\bar{k}]\!].\mathtt{end}) \mid *k.[\![P]\!]\right) \setminus^0 r \setminus^0 k
\end{aligned}$$

By looking at the possible reductions of these (derived) operators, we can define derived reduction rules. Assume δ is the log $[\![\mu_1]\!].\cdots.[\![\mu_n]\!]$, we have a simulation result comparable to the case for multi-synchronization, namely:

$$\begin{aligned}
[\![\left(\mu_1 \times \cdots \times \mu_n\right).P]\!];\sigma \uplus \mathrm{rd}(\delta) &\rightarrow^* \approx_a [\![P]\!];\sigma \uplus \mathrm{wt}(\delta) \\
[\![*\left(\mu_1 \times \cdots \times \mu_n\right).P]\!];\sigma \uplus \mathrm{rd}(\delta) &\rightarrow^* \approx_a [\![*\left(\mu_1 \times \cdots \times \mu_n\right).P]\!] \mid [\![P]\!];\sigma \uplus \mathrm{wt}(\delta)
\end{aligned}$$

To obtain join-definitions, we only need to combine a sequence of replicated multi-synchronizations using the choice composition defined precedently. (We also need hiding to

close the scope of the definition.) Actually, we can encode even more flexible constructs mixing choice and join-patterns. For the sake of simplicity, we only study examples of such operations. The first example is the (linear) join-pattern $(a \times b).P \wedge (a \times c).Q$, that may fire P if the outputs $\{a,b\}$ are in the global state σ and otherwise fire Q if $\{a,c\}$ is in σ (actually, real implementations of join-calculus have a preemptive semantics for pattern synchronization). The second example is the derived operator $(a \times b) + (b \times c \times \overline{a}).P$, such that P is fired if outputs on $\{a,b\}$ are available or if outputs on $\{b,c\}$ are available (in which case an output on a is also generated). These examples can be easily interpreted using atomic transactions:

$$\begin{aligned}
[\![(a \times b).P \wedge (a \times c).Q]\!] &\triangleq \big(\texttt{atom}\big(\ [\![a]\!].[\![b]\!].[\![\overline{k}_1]\!].\texttt{end}\ \texttt{orElse} \\
&\qquad\qquad [\![a]\!].[\![c]\!].[\![\overline{k}_2]\!].\texttt{end}\big)\ \mid\ k_1.P\ \mid\ k_2.Q\big) \setminus^0 k_1 \setminus^0 k_2 \\
[\![\big(a \times b + b \times c \times \overline{a}\big).P]\!] &\triangleq \big(\texttt{atom}\big(\ [\![a]\!].[\![b]\!].[\![\overline{k}]\!].\texttt{end}\ \texttt{orElse} \\
&\qquad\qquad [\![b]\!].[\![c]\!].[\![\overline{a}]\!].[\![\overline{k}]\!].\texttt{end}\big)\ \mid\ k.P\big) \setminus^0 k
\end{aligned}$$

In the next section we define the notion of bisimulation used for reasoning on the soundness of our encodings. We also define an equivalence relation for atomic expressions that is useful for reasoning on the behavior of atomic blocks.

4 Bisimulation Semantics

A first phase before obtaining a bisimulation equivalence is to define a Labeled Transition System (LTS) for AtCCS processes related to the reduction semantics.

Labeled Semantics of AtCCS. It is easy to derive labels from the reduction semantics given in Table 2. For instance, a reduction of the form $P;\sigma \to P';\sigma \uplus \{a\}$ is clearly an *output transition* and we could denote it using the transition $P \xrightarrow{\overline{a}} P'$, meaning that the effect of the transition is to add a message on a to the global state σ. We formalize the notion of label and transition. Besides output actions $\overline{a}$, which corresponds to an application of rule (out), we also need *block actions*, which are multiset $\{a_1,\ldots,a_n\}$ corresponding to the commit of an atomic block, that is to the deletion of a bunch of names from the global state in rule (atOk). Block actions include the usual labels found in LTS for CCS and are used for labeling input and communication transitions: an input actions a, which intuitively corresponds to rules (in) and (rep), is a shorthand for the (singleton) block action $\{a\}$; the silent action τ, which corresponds to rule (com), is a shorthand for the empty block action $\emptyset$. In the following, we use the symbols $\theta,\gamma,\ldots$ to range over block actions and $\mu,\mu',\ldots$ to range over labels, $\mu ::= \overline{a} \mid \theta \mid \tau \mid a$.

The labeled semantics for AtCCS is the smallest relation $P \xrightarrow{\mu} P'$ satisfying the two following clauses:

1. we have $P \xrightarrow{\overline{a}} P'$ if there is a state σ such that $P;\sigma \to P';\sigma \uplus \{a\}$;
2. we have $P \xrightarrow{\theta} P'$ if there is a state σ such that $P;\sigma \uplus \theta \to P';\sigma$.

Note that, in the case of the (derived) action τ, we obtain from clause 2 that $P \xrightarrow{\tau} P'$ if there is a state σ such that $P;\sigma \to P';\sigma$. As usual, silent actions label transitions that do not modify the environment (in our case the global state) and so are invisible to an outside observer. Unlike CCS, the calculus has more examples of silent transition than mere internal synchronization, e.g. the initiation and evolution of an atomic block, see e.g. rules (atST) and (atPass). Consequently, a suitable (weak) equivalence for AtCCS should not distinguish e.g. the processes

$\texttt{atom}(\texttt{retry})$, $\texttt{atom}(\texttt{end})$, $(a.\overline{a})$ and $\mathbf{0}$. The same is true with input transitions. For instance, we expect to equate the processes $a.\mathbf{0}$ and $\texttt{atom}(\texttt{rd}\,a\,.\texttt{end})$.

Our labeled semantics for AtCCS is not based on a set of transition rules, as it is usually the case. Nonetheless, we can recover an axiomatic presentation of the semantics using the tight correspondence between labeled transitions and reductions characterized by Proposition 1.

Proposition 1. *Consider two processes P and Q. The following implications are true:*

(com) *if* $P \xrightarrow{a} P'$ *and* $Q \xrightarrow{\overline{a}} Q'$ *then* $P \mid Q \xrightarrow{\tau} P' \mid Q'$*;*
(par) *if* $P \xrightarrow{\mu} P'$ *then* $P \mid Q \xrightarrow{\mu} P' \mid Q$ *and* $Q \mid P \xrightarrow{\mu} Q \mid P'$*;*
(hid) *if* $P \xrightarrow{\mu} P'$ *and the name a does not appear in* μ *then* $P \setminus^n a \xrightarrow{\mu} P' \setminus^n a$*;*
(hidOut) *if* $P \xrightarrow{\overline{a}} P'$ *then* $P \setminus^n a \xrightarrow{\tau} P' \setminus^{n+1} a$*;*
(hidAt) *if* $P \xrightarrow{\mu} P'$ *and* $\mu = \theta \uplus \{a^m\}$*, where a is a name that does not appear in the label* θ*, then* $P \setminus^{n+m} a \xrightarrow{\theta} P' \setminus^n a$*.*

Proof. In each case, we have a transition of the form $P \xrightarrow{\mu} P'$. By definition, there are states σ and σ' such that $P;\sigma \to P';\sigma'$. The property is obtained by a simple induction on this reduction (a case analysis on the last reduction rule is enough). □

We define additional transition relations used in the remainder of the paper. As usual, we denote by $\Rightarrow$ the *weak transition relation*, that is the reflexive and transitive closure of $\xrightarrow{\tau}$. We denote by $\overset{\mu}{\Rightarrow}$ the relation $\Rightarrow \xrightarrow{\mu} \Rightarrow$. If s is a sequence of labels $\mu_0 \ldots \mu_n$, we denote $\overset{s}{\Rightarrow}$ the relation such that $P \xrightarrow{s} P'$ if and only if there is a process Q such that $P \xrightarrow{\mu_0} Q$ and $Q \xrightarrow{\mu_1 \cdots \mu_n} P'$ (and $\overset{s}{\Rightarrow}$ is the identity relation when s is the empty sequence ε). We also define a weak version $\overset{s}{\Rightarrow}$ of this relation in the same way. Lastly, we denote $\xrightarrow{a^n}$ the relation $\xrightarrow{a} \ldots \xrightarrow{a}$, the composition of n copies of $\xrightarrow{a}$.

Asynchronous Bisimulation for Processes and Expressions. Equipped with a labeled transition system, we can define the traditional (weak) bisimulation equivalence $\approx$ between processes. This is the largest equivalence $\mathcal{R}$ such that if $P\mathcal{R}Q$ and $P \xrightarrow{\mu} P'$ then $Q \overset{\mu}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$. Weak bisimulation can be used to prove interesting equivalences between processes. For instance, we can prove that $(P \mid \overline{a}) \setminus^n a \approx P \setminus^{n+1} a$. Nonetheless, a series of equivalence laws are not valid for $\approx$. For instance, $\texttt{atom}(\texttt{rd}\,a\,.\texttt{end}) \not\approx a.\mathbf{0}$, meaning that, whereas there are no context that separates the two processes, it is possible to test the presence of an atomic block. Also, the usual *asynchronous law* is not valid: $a.\overline{a} \not\approx \mathbf{0}$. To overcome these limitations, we define a weak *asynchronous bisimulation* relation, denoted $\approx_a$, in the style of [1].

Definition 1 (weak asynchronous bisimulation). *A symmetric relation* $\mathcal{R}$ *is a weak asynchronous bisimulation if whenever* $P\mathcal{R}Q$ *then the following holds:*

1. *if* $P \xrightarrow{\overline{a}} P'$ *then there is* Q' *such that* $Q \overset{\overline{a}}{\Rightarrow} Q'$ *and* $P'\mathcal{R}Q'$*;*
2. *if* $P \xrightarrow{\theta} P'$ *then there is a process* Q' *and a block action* γ *such that* $Q \overset{\gamma}{\Rightarrow} Q'$ *and* $\left(P' \mid \prod_{a \in (\gamma \setminus \theta)} \overline{a}\right) \mathcal{R} \left(Q' \mid \prod_{a \in (\theta \setminus \gamma)} \overline{a}\right)$.

We denote with $\approx_a$ *the largest weak asynchronous bisimulation.*

Assume $P \approx_a Q$ and $P \xrightarrow{\tau} P'$, the (derived) case for silent action entails that there is Q' and θ such that $Q \overset{\theta}{\Rightarrow} Q'$ and $P' \mid \prod_{a \in \theta} \overline{a} \approx_a Q'$. If θ is the silent action, $\theta = \{\}$, we recover the usual condition for bisimulation, that is $Q \Rightarrow Q'$ and $P' \approx_a Q'$. If θ is an input action, $\theta = \{a\}$, we recover the definition of asynchronous bisimulation of [1]. Due to the presence of block actions γ, the definition of $\approx_a$ is slightly more complicated than in [1], but it is also more compact (we only have two cases) and more symmetric. Hence, we expect to be able to reuse

known methods and tools for proving the equivalence of AtCCS processes. Another indication that $\approx_a$ is a good choice for reasoning about processes is that it is a congruence. The proof is reported in Appendix A.

Theorem 1. *Weak asynchronous bisimulation $\approx_a$ is a congruence.*

We need to define a specific equivalence relation to reason on transactions. Indeed, the obvious choice that equates two expressions M and N if $\mathtt{atom}(M) \approx_a \mathtt{atom}(N)$ does not lead to a congruence. For instance, we have $(\mathtt{rd}\,a\,.\mathtt{wt}\,a\,.\mathtt{end})$ equivalent to $\mathtt{end}$ while $\mathtt{atom}(\mathtt{rd}\,a\,.\mathtt{wt}\,a\,.\mathtt{end}\ \mathtt{orElse}\ \mathtt{wt}\,b\,.\mathtt{end}) \not\approx_a \mathtt{atom}(\mathtt{end}\ \mathtt{orElse}\ \mathtt{wt}\,b\,.\mathtt{end})$. The first transaction may output a message on b while the second always end silently.

We define an equivalence relation between atomic expressions $\backsimeq$, and a *weak atomic preorder* $\sqsupseteq$, that relates two expressions if they end (or abort) for the same states. We also ask that equivalent expressions should perform the same changes on the global state when they end. We say that two logs δ, δ' have same effects, denoted $\delta =_\sigma \delta'$ if $\sigma \setminus \mathrm{rd}(\delta) \uplus \mathrm{wt}(\delta) = \sigma \setminus \mathrm{rd}(\delta') \uplus \mathrm{wt}(\delta')$. We say that $M \sqsupseteq_\sigma N$ if and only if either (1) $(N)_{\sigma;\varepsilon} \Rightarrow (\mathtt{retry})_{\sigma,\delta}$; or (2) $(N)_{\sigma;\varepsilon} \Rightarrow (\mathtt{end})_{\sigma,\delta}$ and $(M)_{\sigma;\varepsilon} \Rightarrow (\mathtt{end})_{\sigma;\delta'}$. Similarly, we have $M \backsimeq_\sigma N$ if and only if either (1) $(M)_{\sigma;\varepsilon} \Rightarrow (\mathtt{retry})_{\sigma,\delta}$ and $(N)_{\sigma;\varepsilon} \Rightarrow (\mathtt{retry})_{\sigma,\delta'}$; or (2) $(M)_{\sigma;\varepsilon} \Rightarrow (\mathtt{end})_{\sigma;\delta}$ and $(N)_{\sigma;\varepsilon} \Rightarrow (\mathtt{end})_{\sigma,\delta'}$ with $\delta =_\sigma \delta'$.

Definition 2 (weak atomic equivalence). *Two atomic expressions M, N are equivalent, denoted $M \backsimeq N$, if and only if $M \backsimeq_\sigma N$ for every state σ. Similarly, we have $M \sqsupseteq N$ if and only if $M \sqsupseteq_\sigma N$ for every state σ.*

While the definition of $\sqsupseteq$ and $\backsimeq$ depend on a universal quantification over states, testing the equivalence of two expressions is not expensive. First, we can rely on a monotonicity property of reduction: if $\sigma \subseteq \sigma'$ then for all M the effect of $(M)_{\sigma,\delta}$ is included in those of $(M)_{\sigma',\delta}$. Moreover, we define a normal form for expressions later in this section (see Proposition 2) that greatly simplifies the comparison of expressions. Another indication that $\backsimeq$ is a good choice of equivalence for atomic expressions is that it is a congruence. The proof is reported in Appendix A.

Theorem 2. *Weak atomic equivalence $\backsimeq$ is a congruence such that $M \backsimeq N$ implies that for every σ and δ $\{\!|(M)_{\sigma;\delta}|\!\}_M \approx_a \{\!|(N)_{\sigma;\delta}|\!\}_N$ and $\mathtt{atom}(M) \approx_a \mathtt{atom}(N)$.*

Example: Dining Philosophers. In this example we give yet another solution to the well-known dining philosophers problem. We use atomic blocks of actions in the implementation of the system and we show that the obtained process behaves as its specification, without using backtracking and without falling into situations of deadlock. Suppose to have four philosophers, $I = \{0,1,2,3\}$ is the considered set of indexes. In what follows we write $+$ for the sum modulo 4. Suppose t is a set of indexes corresponding to thinking philosophers, which are ready to eat; and e corresponds to eating philosophers, which are ready to think. $P_{t;e}$ is the specification of the system, with $t \cup e = I$, $t \cap e = \emptyset$ and there isn't $i \in I$ such that $i, i+1 \in e$.

$$
\begin{aligned}
P_{t;e} \triangleq\ & \textstyle\sum_{i \notin t} t_i.P_{t \cup i;e-i} \\
& + \textstyle\sum_{\{i=0,1 \text{ if } e=\emptyset\}} \tau.\left(e_i.P_{t-i;i} + e_{i+2}.P_{t-(i+2);(i+2)}\right) \\
& + \textstyle\sum_{\{i \in t \,|\, i-1, i+1 \notin e, i+2 \in e\}} \tau.\, e_i\,.P_{t-i;e \cup i}
\end{aligned}
$$

The system specification will never fall into deadlocks and there can be at most two eating philosophers (with indexes i and $i+2$). The actions of eating and thinking of the philosopher i, e_i and t_i, can be observed as inputs.

A philosopher D_i, for $i \in I$, can be implemented as follows:

$$D_i \triangleq \mathtt{atom}(\mathtt{rd}\, c_{i-1}\, .\mathtt{rd}\, c_i\, .\mathtt{end}).\, e_i\, .\, t_i\, .(\overline{c_{i-1}}\,|\,\overline{c_i}).$$

Process D_i attempts to get the chopsticks, on his right and left, by using an atomic block for reading c_{i-1} and c_i. If the commit of the atomic block can not be performed then at least one of its neighbors, D_{i-1} or D_{i+1} is already eating, because at least one of the chopsticks is not available, thus D_i will retry to get both chopsticks. Otherwise he can eat, thus he will acquire the chopsticks and eat by inputting e_i. After eating, he can decide to start thinking, thus he reads t_i, and after that he releases both chopsticks.

The global system is given by the parallel composition of the philosopher D_i and the output of the four chopsticks, which are hidden to observers

$$D \triangleq (D_0\,|\,D_1\,|\,D_2\,|\,D_3\,|\,\overline{c_0}\,|\,\overline{c_1}\,|\,\overline{c_2}\,|\,\overline{c_3}) \setminus^0 c_0, c_1, c_2, c_3.$$

In what follows we show that $P_{I;\emptyset} \approx_a D$ holds. Before we need to define a useful abbreviation. Suppose $A, B, C, D, E \subseteq \{0, 1, 2, 3\}$, are sets of indexes such that $A \cup B \cup C = \{0, 1, 2, 3\}$, $A \cap B = A \cap C = B \cap C = \emptyset$ and $D \cup E \subseteq \{0, 1, 2, 3\}$ with $D \cap E = \emptyset$. We define $D\{A;B;C;D;E\}$ as follows:

$$\begin{aligned} D\{A;B;C;D;E\} \triangleq\ & (\textstyle\prod_{\{i\in A\}} D_i\,|\,\prod_{\{i\in B\}} e_i.\, t_i\, .(\overline{c_{i-1}}\,|\,\overline{c_i}) \\ & |\,\textstyle\prod_{\{i\in C\}} t_i.(\overline{c_{i-1}}\,|\,\overline{c_i}) \\ & |\,\textstyle\prod_{\{i\in D\}} \overline{c_i}) \setminus^1 c_i\,|\,i \in E \setminus^0 c_i\,|\,i \in D. \end{aligned}$$

That is a system where the philosophers in A are in the initial state; philosophers in B are ready to eat (they have already acquired the chopsticks); philosophers in C are ready to think (they have already eaten); indexes in D correspond to available chopsticks not yet outputted; indexes in E correspond to chopsticks outputted, thus chopsticks that can be taken by some philosopher for eating.

In the following $\mathcal{P}(S)$ represents the powerset of S. $P_{I;\emptyset}\,\mathcal{R}\,D$ where the weak asynchronous bisimulation $\mathcal{R}$ is defined as follows:

$$\begin{aligned} \mathcal{R} = & \big\{(P_{I;\emptyset}, D\{I;\emptyset;\emptyset;I\setminus S;S\})\,|\,S \in \mathcal{P}(I)\big\} \\ & \cup \big\{(P_{I-i;i}, D\{I-i;\emptyset;\{i\};\{i+1,i+2\}\setminus S;S\})\,|\,S \in \mathcal{P}(\{i+1,i+2\})\big\} \\ & \cup \big\{(P_{I-i;i}, D\{\{i-1,i+1\};\{i+2\};\{i\};\emptyset;\emptyset\})\big\} \\ & \cup \big\{(P_{\{i-1,i+1\};\{i,i+2\}}, D\{\{i-1,i+1\};\emptyset;\{i,i+2\};\emptyset;\emptyset\})\big\} \\ & \cup \big\{((e_i.P_{I-i;i} + e_{i+2}.P_{I-(i+2);(i+2)}), D\{\{i-1,i+1\};\{i+2,i\};\emptyset;\emptyset;\emptyset\})\,|\,i=0,1\big\} \\ & \cup \big\{((e_i.P_{I-i;i} + e_{i+2}.P_{I-(i+2);(i+2)}), D\{\{i-1,i,i+1\};\{i+2\};\emptyset;\{i-1,i\}\setminus S;S\}) \\ & \qquad |\,S \in \mathcal{P}(\{i-1,i\}),\, i=0,1\big\} \\ & \cup \big\{((e_i.P_{I-i;i} + e_{i+2}.P_{I-(i+2);(i+2)}), D\{\{i-1,i+1,i+2\};\{i\};\emptyset;\{i+1,i+2\}\setminus S;S\}) \\ & \qquad |\,S \in \mathcal{P}(\{i+1,i+2\}),\, i=0,1\big\}. \end{aligned}$$

On the Algebraic Structure of Transactions. The equivalence relations $\backsimeq$ and $\approx_a$ can be used to prove interesting laws of atomic expressions and processes. We list some of these laws in Table 4. Appropriate bisimulation relations which prove laws in Table 4 are reported in

Laws for atomic expressions:

(comm)	$\alpha.\beta.M$	$\backsimeq$	$\beta.\alpha.M$
(dist)	$\alpha.(M \texttt{ orElse } N)$	$\backsimeq$	$(\alpha.M) \texttt{ orElse } (\alpha.N)$
(ass)	$M_1 \texttt{ orElse } (M_2 \texttt{ orElse } M_3)$	$\backsimeq$	$(M_1 \texttt{ orElse } M_2) \texttt{ orElse } M_3$
(idem)	$M \texttt{ orElse } M$	$\backsimeq$	M
(absRt1)	$\alpha.\texttt{retry}$	$\backsimeq$	$\texttt{retry}$
(absRt2)	$\texttt{retry orElse } M$	$\backsimeq$	$M \backsimeq M \texttt{ orElse retry}$
(absEnd)	$\texttt{end orElse } M$	$\backsimeq$	$\texttt{end}$

Laws for processes:

(asy)	$a.\overline{a}$	$\approx_a$	$\mathbf{0}$
(a-asy)	$\texttt{atom}(\texttt{rd}\,a\,.\texttt{wt}\,a\,.\texttt{end})$	$\approx_a$	$\mathbf{0}$
(a-1)	$\texttt{atom}(\texttt{rd}\,a\,.\texttt{end})$	$\approx_a$	$a.\mathbf{0}$

Table 4. Algebraic Laws of Transactions.

Appendix B. Let $\mathcal{M}$ denotes the set of all atomic expressions. The behavioral rules for atomic expressions are particularly interesting since they exhibit a rich algebraic structure for $\mathcal{M}$. For instance, rules (comm) and (dist) state that action prefix $\alpha.M$ is a commutative operation that distribute over `orElse`. We also have that $(\mathcal{M}, \texttt{orElse}, \texttt{retry})$ is an idempotent semigroup with left identity `retry`, rules (ass), (absRt2) and (idem), and that `end` annihilates $\mathcal{M}$, rule (absEnd). Most of these laws appear in [19] but are not formally proved.

Actually, we can show that the structure of $\mathcal{M}$ is close to that of a bound join-semilattice. We assume unary function symbols $a(\,)$ and $\overline{a}(\,)$ for every name a (a term $\overline{a}(M)$ is intended to represent a prefix $\texttt{wt}\,a\,.M$) and use the symbols $\sqcup, 1, 0$ instead of $\texttt{orElse}, \texttt{end}, \texttt{retry}$. With this presentation, the behavioral laws for atomic expression are almost those of a semilattice. By definition of $\sqsupseteq$, we have that $M \sqcup M' \backsimeq M$ if and only if $M \sqsupseteq M'$ and for all M, N we have $1 \sqsupseteq M \sqcup N \sqsupseteq M \sqsupseteq 0$.

$$\mu(\mu'(M)) \backsimeq \mu'(\mu(M)) \qquad \mu(M \sqcup N) \backsimeq \mu(M) \sqcup \mu(N) \qquad \mu(0) \backsimeq 0$$
$$0 \sqcup M \backsimeq M \backsimeq M \sqcup 0 \qquad 1 \sqcup M \backsimeq 1$$

It is possible to prove other behavioral laws to support our interpretation of `orElse` has a join. However some important properties are missing, most notably, while $\sqcup$ is associative, it is not commutative. For instance, $a(\overline{b}(1)) \sqcup 1 \not\backsimeq 1$ while $1 \backsimeq 1 \sqcup a(\overline{b}(1))$, rule (absEnd). This observation could help improve the design of the transaction language: it will be interesting to enrich the language so that we obtain a real lattice.

Normal Form for Transactions. Next, we show that it is possible to rearrange an atomic expression (using behavioral laws) to put it into a simple *normal form*. This procedure can be understood as a kind of compilation that transform an expression M into a simpler form.

Informally, an atomic expression M is said to be in *normal form* if it does not contain nested `orElse` (all occurrences are at top level) and if there are no redundant branches. A redundant branch is a sequence of actions that will never be executed. For instance, the read actions in $\texttt{rd}\,a\,.\texttt{end}$ are included in $\texttt{rd}\,a\,.\texttt{rd}\,b\,.\texttt{end}$, then the second branch in the compo-

sition $\big(\mathtt{rd}\,a\,.\mathtt{end}\big)\ \mathtt{orElse}\ \big(\mathtt{rd}\,a\,.\mathtt{rd}\,b\,.\mathtt{end}\big)$ is redundant: obviously, if $\mathtt{rd}\,a\,.\mathtt{end}$ fails then $\mathtt{rd}\,a\,.\mathtt{rd}\,b\,.\mathtt{end}$ cannot succeed. We overload the functions defined on logs and write $\mathrm{rd}(M)$ for the (multiset of) names occurring in read actions in M. We define $\mathrm{wt}(M)$ similarly. In what follows, we abbreviate $(M_1\ \mathtt{orElse}\ \ldots\ \mathtt{orElse}\ M_n)$ with the expression $\bigsqcup_{i\in 1..n} M_i$. We say that an expression M is in *normal form* if it is of the form $\bigsqcup_{i\in 1..n} K_i$ where for all indexes $i,j \in 1..n$ we have: (1) K_i is a sequence of action prefixes $\alpha_{j_1}.\ldots.\alpha_{j_{n_i}}.\mathtt{end}$; and (2) $\mathrm{rd}(K_i) \not\subseteq \mathrm{rd}(K_j)$ for all $i<j$. Condition (1) requires the absence of nested $\mathtt{orElse}$ and condition (2) prohibits redundant branches (it also means that all branches, but the last one, has a read action). The following proposition is proved in Appendix C.

Proposition 2. *For every expression M there is a normal form M' such that $M \backsimeq M'$.*

Our choice of using bisimulation for reasoning about atomic transactions may appear arbitrary. We have already debated over the need to consider asynchronous bisimulation $\approx_a$ instead of (the simple) bisimulation $\approx$. In the next section, we study a testing equivalence for AtCCS, more particularly an asynchronous may testing semantics [17].

5 May-testing semantics

Using a testing equivalence instead of bisimulation is sometimes more convenient. Nonetheless, testing equivalences have the drawback that their definition depends on a universal quantification over arbitrarily many processes. We define a may-testing equivalence for AtCCS and give an alternative characterization using a trace-based equivalence. We also expose some shortcomings of may testing related to the (folklore) fact that it cannot distinguish the points of choice in a process. Actually, we define for every atomic block $\mathtt{atom}(M)$ a corresponding process without transactions (but using choice) that is indistinguishable from $\mathtt{atom}(M)$. The results enunciated in this section are proved in Appendix D.

We define the notion of observers and successful computations. An *observer O* is a particular type of process which does not contain atomic blocks and that can perform a distinct output $\overline{w}$ (the success action). We denote *Obs* the set of all observers. A *computation* from a process P and an observer O is a sequence of transitions of the form $P \mid O = P_0 \mid O_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_k \mid O_k \xrightarrow{\tau} \ldots$, which is either infinite or of finite size, say n, such that $P_n \mid O_n$ cannot evolve. A computation from $P \mid O$ is *successful* if there is an index n such that O_n has a success action, that is $O_n \xrightarrow{\overline{w}}$. In this case, we say that *P may O*. Two processes are may testing equivalent if they have the same successful observers.

Definition 3 (may-testing preorder). *Given two processes P and Q, we write $P \sqsubseteq_{may} Q$ if for every observer O in Obs we have P may O implies Q may O. We use $\simeq_{may}$ to denote the equivalence obtained as the kernel of the preorder $\sqsubseteq_{may}$.*

Universal quantification on observers make it difficult to work with the operational definition of the may preorder. Following [7], we study a trace-based characterization for our calculus. The following preorder over traces will be used for defining the alternative characterization of the may-testing preorder.

In our setting, a *trace s* is a sequence of actions $\mu_1 \ldots \mu_n$. (We only consider output and block actions and leave aside τ and input actions, which are derivable). We define a preorder $\preceq_0$ on traces as the smallest relation that satisfies the following laws.

$$\text{(TO1) } s_1\, s_2 \ \preceq_0\ s_1\,\{a\}\,s_2 \qquad \text{(TO2)}\quad s_1\, s_2\,\{a\}\,s_3 \ \preceq_0\ s_1\,\{a\}\,s_2\,s_3$$

$$\text{(TO3) } s_1\, s_2 \ \preceq_0\ s_1\,\{a\}\,\overline{a}\,s_2 \qquad \text{(TO4) } \{a_1,\ldots,a_n\} \ {}_0\!\succeq \preceq_0\ \{a_1\}\ldots\{a_n\}$$

Following the terminology of [7], (TO1), (TO2) and (TO3) are the laws for *deletion*, *postponement* and *annihilation* of input action. We add rule (TO4) which allows to substitute block actions with the corresponding sequences of inputs. The simulation relation $\preceq$ is the reflexive and transitive closure of $\preceq_0$. The preorder $\preceq$ is preserved by prefixing. We can now define a preorder over processes.

Definition 4 (alternative preorder). *For processes P and Q, we set $P \ll_{may} Q$ if for all weak transition $P \stackrel{s}{\Rightarrow} P'$ there is a trace s' and a process Q' such that $s' \preceq s$ and $Q \stackrel{s'}{\Rightarrow} Q'$.*

We now prove coincidence of $\ll_{may}$ and $\sqsubseteq_{may}$. Some definitions and preliminary results are needed. For every label μ we define the complement $\overline{\mu}$ such that: the complement of an output action $\overline{a}$ is a block action $\{a\}$ and the complement of a block action $\{a_1,\ldots,a_n\}$ is a trace $\overline{a}_1 \ldots \overline{a}_n$. For every trace $s = \mu_1 \ldots \mu_n$, the cotrace $\overline{s} = \overline{\mu}_1 \ldots \overline{\mu}_n$ is obtained by concatenating the complements of the actions in s. The following lemma relates the preorder $\preceq$ with the operational semantics of processes.

Lemma 1. *Assume that $s' \preceq s$ and $P \stackrel{\overline{s}}{\Rightarrow} P'$, then there is a process P'' such that $P \stackrel{\overline{s}'}{\Rightarrow} P''$.*

The next step is to define a special class of observers. For every trace s, we inductively define an observer $O(s) \in Obs$ as follows:

$$O(\varepsilon) \stackrel{\triangle}{=} \overline{w}, \quad O(\overline{a}\,s) \stackrel{\triangle}{=} a.O(s), \quad O(\{a_1,\ldots,a_n\}\,s) \stackrel{\triangle}{=} \big(\prod_{i\in 1..n} \overline{a_i}\big) \mid O(s)$$

The following property shows that the sequence of visible actions from $O(s)$ is related to traces simulated by s.

Lemma 2. *Consider two traces s and r. If there is a process Q such that $O(s) \stackrel{\overline{r}}{\Rightarrow} \stackrel{\overline{w}}{\Rightarrow} Q$ then $r \preceq s$.*

We can now prove a full abstraction theorem between may testing $\sqsubseteq_{may}$ and the alternative preorder $\ll_{may}$.

Theorem 3. *For all processes P and Q, we have $P \sqsubseteq_{may} Q$ if and only if $P \ll_{may} Q$.*

Next, we show that may-testing semantics is not precise enough to tell apart atomic transactions from sequences of input actions. We consider an atomic expression M in normal form. Assume $M = \bigsqcup_{i\in 1..n} M_i$, the following lemma state that the observing behavior of M is obtained by considering, for every branch K_i, a transition labeled by the block action containing $\mathrm{rd}(K_i)$ followed by output transitions on the names in $\mathrm{wt}(K_i)$.

Lemma 3. *Assume $M = \bigsqcup_{i\in 1..n} K_i$ is an expression in normal form. For every index i in $\{1,\ldots,n\}$ we have $\mathtt{atom}(M);\sigma_i \rightarrow^* \{\!|(\mathtt{end})_{\sigma_i;\delta}|\!\}_M;\sigma_i$ where $\sigma_i = \mathrm{rd}(K_i) = \mathrm{rd}(\delta)$ and $\mathrm{wt}(\delta) = \mathrm{wt}(K_i)$.*

As a corollary of Lemma 3, we obtain that the possible behavior of $\mathtt{atom}(M)$ can be described as $\mathtt{atom}(M) \stackrel{\sigma_i}{\Rightarrow} \prod_{b\in \mathrm{wt}(K_i)} \overline{b}$ for every $i \in 1..n$, where σ_i is the multiset $\mathrm{rd}(K_i)$.

We now prove that for every atomic transaction $\mathtt{atom}(M)$ there is a CCS process $[\![M]\!]$ that is may-testing equivalent to M. By CCS process, we intend a term of AtCCS without atomic transaction that may include occurrences of the choice operator $P+Q$. By Proposition 2, we can assume that M is in normal form, that is $M = \bigsqcup_{i\in 1..n} K_i$. The interpretation of a sequence of actions $K = \alpha_1.\ldots.\alpha_n.\mathtt{end}$ is the process $[\![K]\!] = a_1.\cdots.a_k.\big(\overline{b_1} \mid \cdots \mid \overline{b_l}\big)$ where $\{a_1,\ldots,a_k\} =$

$\mathrm{rd}(K)$ and $\{b_1,\ldots,b_l\} = \mathrm{wt}(K)$. (In particular we have $[\![\mathtt{end}]\!] = \mathbf{0}$.) The translated of M, denoted $[\![M]\!]$, is the process $[\![K_1]\!] + \cdots + [\![K_n]\!]$. The following theorem proves that may-testing semantics is not able to distinguish the behavior of an atomic process from the behavior of its translation, which means that may-testing is blind to the presence of transactions.

Proposition 3. *For every expression M in normal form we have* $\mathtt{atom}(M) \simeq_{may} [\![M]\!]$.

We observe that a process $[\![M]\!]$ is a choice between processes of the form $a.P$ or $\left(\prod_{i\in I} \overline{b_i}\right)$. Therefore, using internal choice and a slightly more convoluted encoding, it is possible to use only input guarded choice $a.P + b.Q$ in place of full choice in the definition of $[\![M]\!]$.

6 Future and Related Works

There is a long history of works that try to formalize the notions of transactions and atomicity, and a variety of approaches to tackle this problem. We review some of these works that are the most related to ours.

We can list several works that combine ACID transactions with process calculi. Gorrieri et al [18] have modeled concurrent systems with atomic behaviors using an extension of CCS. They use a two-level transition systems (a high and a low level) where high actions are decomposed into atomic sequences of low actions. To enforce isolation, atomic sequences must go into a special invisible state during all their execution. Contrary to our model, this work does not follow an optimistic approach: sequences are executed sequentially, without interleaving with other actions, as though in a critical section. Another related calculus is RCCS, a reversible version of CCS [15,16] based on an earlier notion of process calculus with backtracking [3]. In RCCS,each process has access to a log of its synchronization's history and may always wind back to a previous state. This calculus guarantees the ACD properties of transactions (isolation is meaningless since RCCS do not use a shared memory model). Finally, a framework for specifying the semantics of transactions in an object calculus is given in [26]. The framework is parametrized by the definition of a transactional mechanism and allows the study of multiple models, such as e.g. the usual lock-based approach. In this work, STM is close to a model called *versioning semantics*. Like in our approach, this model is based on the use of logs and is characterized by an optimistic approach where log consistency is checked at commit time. Fewer works consider behavioral equivalences for transactions. A foundational work is [5], that gives a theory of transactions specifying atomicity, isolation and durability in the form of an equivalence relation on processes, but it provides no formal proof system.

Linked to the upsurge of works on Web Services (and on long running Web transactions), a larger body of works is concerned with formalizing *compensating transactions*. In this context, each transactive block of actions is associated with a compensation (code) that has to be run if a failure is detected. The purpose of compensation is to undo most of the visible actions that have been performed and, in this case, atomicity, isolation and durability are obviously violated. We give a brief survey of works that formalize compensable processes using process calculi. These works are of two types: (1) *interaction based compensation* [6,8,21], which are extensions of process calculi (like π or join-calculus) for describing transactional choreographies where composition take place dynamically and where each service describes its possible interactions and compensations; (2) *compensable flow composition* [9,11,12,13], where ad hoc process algebras are designed from scratch to describe the possible flow of control among services. These calculi are oriented towards the orchestration of services and service failures. This second approach is also followed in [2,4] where two frameworks for composing transactional services are presented.

The study of AtCCS is motivated by our objective to better understand the semantics of the STM model. Obtaining a suitable behavioral equivalence for atomic expression is a progress for the verification of concurrent applications that use STM. However, we can imagine using our calculus for other purposes. An interesting problem is to develop an approach merging atomic and compensating transactions. A first step in this direction is to enrich our language and allow the parallel composition of atomic expressions and the nesting of transactions. We are currently working on this problem. Another area for research stems from our observation (see Section 4) that the algebraic structure of atomic expressions is lacking interesting property. Indeed, it will be interesting to enrich the language of expressions in order to obtain a real lattice. The addition of a symmetric choice operator for atomic expressions may be a solution, but it could introduce unwanted nondeterminism in the evaluation of transactions.

References

1. R. Amadio, I. Castellani and D. Sangiorgi. On Bisimulations for the Asynchronous π-Calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.
2. D. Berardi, D. Calvanese, G. De Giacomo, R. Hull and M. Mecella. Automatic Composition of Transition-Based Web Services with Messaging. In *Proc. of VLDB*, 2005.
3. J.A. Bergstra, A. Ponse and J.J. van Wamel. Process Algebra with Backtracking. In *Proc. of REX Workshop*, LNCS 803, 1994.
4. S. Bhiri, O. Perrin and C. Godart. Ensuring Required Failure Atomicity of Composite Web Services. In *Proc. of WWW*, ACM Press, 2005.
5. A.P. Black, V. Cremet, R. Guerraoui and M. Odersky. An Equational Theory for Transactions. In *Proc. of FSTTCS*, LNCS 2914, 2003.
6. L. Bocchi, C. Laneve and G. Zavattaro. A Calculus for Long Running Transactions. In *Proc. of FMOODS*, LNCS 2884, 2003.
7. M. Boreale, R. De Nicola and R. Pugliese. Trace and Testing Equivalence on Asynchronous Processes. *Information and Computation*, 172(2): 139–164, 2002.
8. R. Bruni, H.C. Melgratti and U. Montanari. Nested Commits for Mobile Calculi: extending Join. In *Proc. of IFIP TCS*, 563–576, 2004.
9. R. Bruni, H.C. Melgratti and U. Montanari. Theoretical Foundations for Compensations in Flow Composition Languages. In *Proc. of POPL*, ACM Press, 209–220, 2005.
10. N. Busi, R. Gorrieri, G. Zavattaro. A Process Algebraic View of Linda Coordination Primitives. *Theoretical Computer Science*, 192(2):167–199, 1998.
11. M.J. Butler, C.A.R. Hoare and C. Ferreira. A Trace Semantics for Long-Running Transactions. In *Proc. of 25 Years Communicating Sequential Processes*, LNCS 3525, 2004.
12. M.J. Butler, C.A.R. Hoare and C. Ferreira. An Operational Semantics for StAC, a Language for Modeling Long Running Transactions. In *Proc. of COORDINATION*, LNCS 2949, 2004.
13. M.J. Butler, C. Ferreira and M.Y. Ng. Precise Modeling of Compensating Business Transactions and its Application to BPEL. In *J. UCS*, 11:712–743, 2005.
14. T. Chothia and D. Duggan. Abstractions for Fault-Tolerant Global Computing. *Theoretical Computer Science*, 322(3):567–613, 2004.
15. V. Danos and J. Krivine. Reversible Communicating System. In *Proc. of CONCUR*, LNCS 3170, 2004.
16. V. Danos and J. Krivine. Transactions in RCCS. In *Proc. of CONCUR*, LNCS 3653, 2005.
17. R. De Nicola and M.C.B. Hennessy. Testing Equivalence for Processes. *Theoretical Computer Science*, 34:83–133, 1984.
18. R. Gorrieri, S. Marchetti and U. Montanari. A^2CCS: Atomic Actions for CCS. *Theoretical Computer Science*, 72(2-3):203–223, 1990.
19. T. Harris, S. Marlow, S.P. Jones and M. Herlihy. Composable Memory Transactions. In *Proc. of PPOPP*, ACM Press, 48–60, 2005.

20. M. Herlihy, J.E. Moss. Transactional Memory: Architectural Support for Lock-Free Data Structures In *Proc. of International Symposium on Computer Architecture*, 1993.
21. C. Laneve and G. Zavattaro. Foundations of Web Transactions. In *Proc. of FoSSaCS*, LNCS 3441, 2005.
22. R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.
23. R. Milner. A Calculus of Communicating Systems. Springer-Verlag, 1982.
24. C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous pi-calculus. Mathematical Structures in Computer Science, 13(5), 2003.
25. N. Shavit and D. Touitou. Software Transactional Memory. In *Proc. of Principles of Distributed Computing*, ACM Press, 1995.
26. J. Vitek, S. Jagannathan, A. Welc and A.L. Hosking. A semantic Framework for Designer Transactions. In *Proc. of ESOP*, LNCS 2986, 2004.

A Proofs of Section 4

Before proving the validity of Theorem 1 and Theorem 2, it is necessary to present some preliminary results.

The following proposition reminds an important property of asynchronous calculi: no behavior causally depends on the execution of output actions. Relation $\sim$ stands for the usual strong bisimulation relation (see e.g. [23]).

Proposition A1 *$P \xrightarrow{\overline{a}} P'$ implies $P \sim P' \,|\, \overline{a}$.*

Proof. By observing that outputs are non-blocking actions, a suitable strong bisimulation can be defined. □

As direct consequences of the previous proposition, we get the results enunciated in the following lemma: (1) output actions can always be delayed and (2) a diamond property involving outputs.

Lemma A1 *Let μ be a generic action ($\mu ::= \overline{b} \,|\, \theta \,|\, \tau$):*

1. *$P \xrightarrow{\overline{a}} \xrightarrow{\mu} P'$ implies $P \xrightarrow{\mu} \xrightarrow{\overline{a}} P'$; similarly $P \xrightarrow{\overline{a}} \stackrel{\mu}{\Rightarrow} P'$ implies $P \stackrel{\mu}{\Rightarrow} \xrightarrow{\overline{a}} P'$;*
2. *$P \xrightarrow{\overline{a}} P'$ and $P \xrightarrow{\mu} P''$ imply that there is a P''' such that $P' \xrightarrow{\mu} P'''$ and $P'' \xrightarrow{\overline{a}} P'''$; similarly $P \xrightarrow{\overline{a}} P'$ and $P \stackrel{\mu}{\Rightarrow} P''$ imply that there is a P''' such that $P' \stackrel{\mu}{\Rightarrow} P'''$ and $P'' \xrightarrow{\overline{a}} P'''$.*

Proof. By Proposition A1. □

The following propositions enunciate two relevant properties of the hiding operator.

Proposition A2 *$(P \,|\, \overline{a}) \setminus^n b \approx_a (P \setminus^n b \,|\, \overline{a})$ if $a \neq b$.*

Proof. By Proposition 1 (hid), and definition of $\xrightarrow{\mu}$. □

Proposition A3 *$(P \,|\, \overline{a}) \setminus^n a \approx_a P \setminus^{n+1} a$.*

Proof. It is enough to note that $(P \,|\, \overline{a}) \setminus^n a \xrightarrow{\tau} P \setminus^{n+1} a$, Proposition 1 (hidAt). □

In the following propositions we prove that $\approx_a$ and $\backsimeq$ are closed under contexts; as a consequence we obtain that both are congruences.

Proposition A4 *$P \approx_a Q$ implies $\forall a : \; a.P \approx_a a.Q$.*

Proof. It is enough to show that the relation $\mathcal{R} = \approx_a \cup \{(a.P, a.Q)\}$ is a weak asynchronous bisimulation. □

Proposition A5 *$P \approx_a Q$ implies $\forall a : \; *a.P \approx_a *a.Q$.*

Proof. It is enough to show that the relation

$$\mathcal{R} = \{((\prod_i P_i^{n_i} \mid *a.P), (\prod_i Q_i^{n_i} \mid *a.Q)) \;\big|\; n_i \geq 0, \; (P_i, Q_i) \in \approx_a\}$$

where P^n is a shorthand for the parallel composition of n copies of P and $\prod_i P_i$ stands for $P_1 | \cdots | P_n | \cdots$, is a weak asynchronous bisimulation up to $\sim$.

The proof proceeds as usual, by showing that every transition of the left term can be matched by a transition of the right one (and vice-versa), and the pair composed by the arrival processes is in $\mathcal{R}$. The proof is straightforward by a simple case analysis of transitions, as defined in Proposition 1. The most involved case is when a communication occurs between two subprocesses, let's say P_j and P_k. Suppose $P_j \xrightarrow{a} P'_j$ and $P_k \xrightarrow{\overline{a}} P'_k$. This means that, by Proposition 1 (com):

$$(\prod_i P_i^{n_i} \mid *a.P) \xrightarrow{\tau} (\prod_{i\neq j,k} P_i^{n_i} \mid P_j^{n_j-1} \mid P_k^{n_k-1} \mid P'_j \mid P'_k \mid *a.P) = R_1 \ .$$

By $P_k \approx_a Q_k$ we know that $Q_k \stackrel{\overline{a}}{\Rightarrow} Q'_k$ with $P'_k \approx_a Q'_k$. We distinguish the following cases for Q_j:

$Q_j \stackrel{a}{\Rightarrow} Q'_j$**:** in this case $Q'_j \approx_a P'_j$ and, by Proposition 1 (com):

$$(\prod_i Q_i^{n_i} \mid *a.Q) \stackrel{\tau}{\Rightarrow} (\prod_{i\neq j,k} Q_i^{n_i} \mid Q_j^{n_j-1} \mid Q_k^{n_k-1} \mid Q'_j \mid Q'_k \mid *a.Q) = R_2$$

and $(R_1, R_2) \in \mathcal{R}$ by definition of $\mathcal{R}$.

$Q_j \stackrel{\theta}{\Rightarrow} Q'_j$**:** this means that, by Proposition 1 (par):

$$(\prod_i Q_i^{n_i} \mid *a.Q) \stackrel{\theta}{\Rightarrow} (\prod_{i\neq j} Q_i^{n_i} \mid Q_j^{n_j-1} \mid Q'_j \mid *a.Q) = R_2$$

and we have to show that $R_1 \mid \prod_{b\in\theta} \overline{b} \approx_a R_2$. We distinguish two cases:

$a \in \theta$**:** from $P_j \approx_a Q_j$ we obtain that $P'_j \mid \prod_{b\in\theta\setminus a} \overline{b} \approx_a Q'_j$. Moreover, remembering that $P'_k \approx_a Q'_k$, we have (by definition of $\mathcal{R}$):

$$(\prod_{i\neq j,k} P_i^{n_i} \mid P_j^{n_j-1} \mid P_k^{n_k-1} \mid P'_j \mid \prod_{b\in\theta\setminus a} \overline{b} \mid P'_k \mid *a.P) \mathcal{R} (\prod_{i\neq j,k} Q_i^{n_i} \mid Q_j^{n_j-1} \mid Q_k^{n_k-1} \mid Q'_j \mid Q'_k \mid *a.Q)$$

but $\overline{a} \approx_a \overline{a}$, thus we also have (again by definition of $\mathcal{R}$)

$$(\prod_{i\neq j,k} P_i^{n_i} \mid P_j^{n_j-1} \mid P_k^{n_k-1} \mid P'_j \mid \prod_{b\in\theta\setminus a} \overline{b} \mid P'_k \mid \overline{a} \mid *a.P) \mathcal{R} (\prod_{i\neq j,k} Q_i^{n_i} \mid Q_j^{n_j-1} \mid Q_k^{n_k-1} \mid Q'_j \mid Q'_k \mid \overline{a} \mid *a.Q)$$

by Proposition A1, $\overline{a} \mid Q'_k \sim Q_k$, thus

$$(\prod_{i\neq j,k} P_i^{n_i} \mid P_j^{n_j-1} \mid P_k^{n_k-1} \mid P'_j \mid \prod_{b\in\theta} \overline{b} \mid P'_k \mid *a.P) \mathcal{R} \sim (\prod_{i\neq j} Q_i^{n_i} \mid Q_j^{n_j-1} \mid Q'_j \mid *a.Q) \ .$$

$a \notin \theta$**:** from $P_j \approx_a Q_j$ we obtain that $P'_j \mid \prod_{b\in\theta} \overline{b} \approx_a Q'_j \mid \overline{a}$. Moreover, remembering that $P'_k \approx_a Q'_k$, we have (by definition of $\mathcal{R}$):

$$(\prod_{i\neq j,k} P_i^{n_i} \mid P_j^{n_j-1} \mid P_k^{n_k-1} \mid P'_j \mid \prod_{b\in\theta} \overline{b} \mid P'_k \mid *a.P) \mathcal{R} (\prod_{i\neq j,k} Q_i^{n_i} \mid Q_j^{n_j-1} \mid Q_k^{n_k-1} \mid Q'_j \mid \overline{a} \mid Q'_k \mid *a.Q)$$

by Proposition A1, $\overline{a} \mid Q'_k \sim Q_k$, thus

$$(\prod_{i\neq j,k} P_i^{n_i} \mid P_j^{n_j-1} \mid P_k^{n_k-1} \mid P'_j \mid \prod_{b\in\theta} \overline{b} \mid P'_k \mid *a.P) \mathcal{R} \sim (\prod_{i\neq j} Q_i^{n_i} \mid Q_j^{n_j-1} \mid Q'_j \mid *a.Q) \ .$$

□

Proposition A6 *$P \approx_a Q$ implies $\forall R:\ P\,|\,R \approx_a Q\,|\,R$.*

Proof. The proof proceeds by showing that the relation

$$\mathcal{R} = \{(P\,|\,R, Q\,|\,R)\,|\,(P,Q) \in \approx_a\}$$

is a weak asynchronous bisimulation up to $\sim$.

Suppose $P\,|\,R \xrightarrow{\mu} S$; by applying Proposition 1, we can distinguish the following cases obtained by applying Proposition 1 (par) or (com):

- $R \xrightarrow{\mu} R'$**:** $S = P\,|\,R'$; by Proposition 1 (par), $Q\,|\,R \xrightarrow{\mu} Q\,|\,R'$ and $(P\,|\,R')\mathcal{R}(Q\,|\,R')$ by definition of $\mathcal{R}$;
- $P \xrightarrow{\overline{a}} P'$**:** $\mu = \overline{a}$ and $S = P'\,|\,R$. By $P \approx_a Q$ we have $Q \stackrel{\overline{a}}{\Rightarrow} Q'$ with $P' \approx_a Q'$. By Proposition 1 (par), $Q\,|\,R \stackrel{\overline{a}}{\Rightarrow} Q'\,|\,R$ and $(P'\,|\,R)\mathcal{R}(Q'\,|\,R)$ by definition of $\mathcal{R}$;
- $P \xrightarrow{\theta} P'$**:** $\mu = \theta$ and $S = P'\,|\,R$. By $P \approx_a Q$ we have $Q \stackrel{\theta'}{\Rightarrow} Q'$ and $\left(P' \,|\, \prod_{a\in\theta'\setminus\theta} \overline{a}\right) \approx_a \left(Q' \,|\, \prod_{a\in\theta\setminus\theta'} \overline{a}\right)$.
  By Proposition 1 (par), $Q\,|\,R \stackrel{\theta'}{\Rightarrow} Q'\,|\,R$ and $\left(P' \,|\, \prod_{a\in\theta'\setminus\theta} \overline{a}\,|\,R\right)\mathcal{R}\left(Q' \,|\, \prod_{a\in\theta\setminus\theta'} \overline{a}\,|\,R\right)$ follows from $\left(P' \,|\, \prod_{a\in\theta'\setminus\theta} \overline{a}\right) \approx_a \left(Q' \,|\, \prod_{a\in\theta\setminus\theta'} \overline{a}\right)$ and definition of $\mathcal{R}$;
- $P \xrightarrow{\overline{a}} P'$ **and** $R \xrightarrow{a} R'$**:** $\mu = \tau$ and $S = P'\,|\,R'$. $P \approx_a Q$ implies $Q \stackrel{\overline{a}}{\Rightarrow} Q'$ and $P' \approx_a Q'$. By Proposition 1 (com), $Q\,|\,R \Rightarrow Q'\,|\,R'$ and, by definition of $\mathcal{R}$, $(P'\,|\,R')\mathcal{R}(Q'\,|\,R')$;
- $P \xrightarrow{\{a\}} P'$ **and** $R \xrightarrow{\overline{a}} R'$**:** $\mu = \tau$ and $S = P'\,|\,R'$. $P \approx_a Q$ implies that $Q \stackrel{\theta}{\Rightarrow} Q'$. We consider the following cases by distinguishing the possible values of θ:
  - $\theta = \{a\}$**:** in this case $P' \approx_a Q'$. By Proposition 1 (com), $Q\,|\,R \Rightarrow Q'\,|\,R'$ and, by definition of $\mathcal{R}$, $(P'\,|\,R')\mathcal{R}(Q'\,|\,R')$;
  - **otherwise:** $Q\,|\,R \stackrel{\theta}{\Rightarrow} Q'\,|\,R$ by Proposition 1 (par); we have to prove that $P'\,|\,R'\,|\,\prod_{b\in\theta}\overline{b} \approx Q'\,|\,R$. We distinguish the following cases:
    $a \in \theta$**:** from $P \approx_a Q$ we obtain $P'\,|\,\prod_{b\in\theta\setminus a}\overline{b} \approx_a Q'$, by definition of $\mathcal{R}$:

$$P' \,|\, \prod_{b\in\theta\setminus a} \overline{b}\,|\,R \approx Q'\,|\,R$$

and by Proposition A1, $R \sim R'\,|\,\overline{a}$, thus

$$P'\,|\,R'\,|\,\prod_{b\in\theta}\overline{b} \sim\approx_a Q'\,|\,R\,;$$

$a \notin \theta$**:** from $P \approx_a Q$ we obtain $P'\,|\,\prod_{b\in\theta}\overline{b} \approx_a Q'\,|\,\overline{a}$, by definition of $\mathcal{R}$:

$$P' \,|\, \prod_{b\in\theta} \overline{b}\,|\,R' \approx_a Q' \,|\, \overline{a}\,|\,R'$$

and by Proposition A1, $R \sim R'\,|\,\overline{a}$, thus

$$P'\,|\,R'\,|\,\prod_{b\in\theta}\overline{b} \approx_a\sim Q'\,|\,R\,.$$

□

Proposition A7 *$P \approx_a Q$ implies $\forall a, n \geq 0:\ P\setminus^n a \approx_a Q\setminus^n a$.*

Proof. The proof proceeds by showing that the relation:

$$\mathcal{R} = \{(P_i\setminus^{n+i} a, Q_j\setminus^{n+j} a)\,|\,n \geq 0,\ (P,Q) \in \approx_a,\ P \xrightarrow{\overline{a}^i} P_i,\ Q \xrightarrow{\overline{a}^j} Q_j\}$$

is a weak asynchronous bisimulation up to $\sim$. We distinguish the following cases:

(hid): $P_i \setminus^{n+i} a \xrightarrow{\mu} P_i' \setminus^{n+i} a$ is derived by $P_i \xrightarrow{\mu} P_i'$, if a not appears in μ. By Lemma A1 (1), $P \xrightarrow{\overline{a}^i} P_i \xrightarrow{\mu} P_i'$ implies $P \xrightarrow{\mu} P' \xrightarrow{\overline{a}^i} P_i'$. From $P \approx_a Q$ we obtain $Q \xRightarrow{\mu} Q'$ with $P' \approx_a Q'$ and by $Q \xrightarrow{\overline{a}^j} Q_j$ and Lemma A1 (2), $Q' \xrightarrow{\overline{a}^j} Q_j'$ and $Q_j \xRightarrow{\mu} Q_j'$; by Proposition 1 (hid), $Q_j \setminus^{n+j} a \xRightarrow{\mu} Q_j' \setminus^{n+j} a$. Finally, $(P_i' \setminus^{n+i} a) \mathcal{R} (Q_j' \setminus^{n+j} a)$ because of $P' \approx_a Q'$, $P' \xrightarrow{\overline{a}^i} P_i'$, $Q' \xrightarrow{\overline{a}^j} Q_j'$ and definition of $\mathcal{R}$;

(hidAt): $P_i \setminus^{n+i} a \xrightarrow{\theta} P_i' \setminus^{n'} a$ is derived by $P_i \xrightarrow{\theta'} P_i'$ with $\theta' = \theta \uplus a^m$ and $n' = n + i - m$. By Lemma A1 (1), $P \xrightarrow{\overline{a}^i} P_i \xrightarrow{\theta} P_i'$ implies $P \xrightarrow{\theta} P' \xrightarrow{\overline{a}^i} P_i'$. By $P \approx_a Q$, $Q \xRightarrow{\gamma} Q'$ with $(P' \mid \prod_{b \in \gamma \setminus \theta'} \overline{b}) \approx_a (Q' \mid \prod_{b \in \theta' \setminus \gamma} \overline{b})$. Suppose $\gamma' = \gamma \uplus a^{m'}$ and, without loss of generality, that $m' > m$. We can rewrite $P' \mid \prod_{b \in \gamma' \setminus \theta'} \overline{b}$ as $P' \mid \overline{a}^{m'-m} \mid \prod_{b \in \gamma \setminus \theta} \overline{b}$ and $Q' \mid \prod_{b \in \theta' \setminus \gamma} \overline{b}$ as $Q' \mid \prod_{b \in \theta \setminus \gamma} \overline{b}$, thus

$$(P' \mid \overline{a}^{m'-m} \mid \prod_{b \in \gamma \setminus \theta} \overline{b}) \approx_a (Q' \mid \prod_{b \in \theta \setminus \gamma} \overline{b}).$$

Moreover, by Lemma A1 (2), $Q \xRightarrow{\gamma} Q'$ and $Q \xrightarrow{\overline{a}^j} Q_j$ imply $Q_j \xRightarrow{\gamma} Q_j'$ and $Q' \xrightarrow{\overline{a}^j} Q_j'$; by Proposition 1 (hidAt), $Q_j \setminus^{n+j} a \xRightarrow{\gamma} Q_j' \setminus^{n+j-m'} a$.

We have to relate the processes $P_i' \setminus^{n+i-m} a \mid \prod_{b \in \gamma \setminus \theta} \overline{b}$ and $Q_j' \setminus^{n+j-m'} a \mid \prod_{b \in \theta \setminus \gamma} \overline{b}$.

By Proposition 1 (hidOut), $(P' \mid \overline{a}^{m'-m} \mid \prod_{b \in \gamma \setminus \theta} \overline{b}) \setminus^{n-m'} a \xrightarrow{\tau} (P_i' \mid \prod_{b \in \gamma \setminus \theta} \overline{b}) \setminus^{n+i-m} a$ and $(Q' \mid \prod_{b \in \theta \setminus \gamma} \overline{b}) \setminus^{n-m'} a \xrightarrow{\tau} (Q_j' \mid \prod_{b \in \theta \setminus \gamma} \overline{b}) \setminus^{n+j-m'} a$; thus from $(P' \mid \overline{a}^{m'-m} \mid \prod_{b \in \gamma \setminus \theta} \overline{b}) \approx_a (Q' \mid \prod_{b \in \theta \setminus \gamma} \overline{b})$ we obtain $\big((P_i' \mid \prod_{b \in \gamma \setminus \theta} \overline{b}) \setminus^{n+i-m} a\big) \mathcal{R} \big((Q_j' \mid \prod_{b \in \theta \setminus \gamma} \overline{b}) \setminus^{n+j-m'} a\big)$, that is $P_i' \setminus^{n+i-m} a \mid \prod_{b \in \gamma \setminus \theta} \overline{b} \sim \mathcal{R} \sim Q_j' \setminus^{n+j-m'} a \mid \prod_{b \in \theta \setminus \gamma} \overline{b}$, by Proposition A2.

(hidOut): $P_i \setminus^{n+i} a \xrightarrow{\tau} P_i' \setminus^{n+i+1} a$ is derived by $P_i \xrightarrow{\overline{a}} P_i'$; $P_i' = P_{i+1}$ and by definition of $\mathcal{R}$ we have $(P_{i+1} \setminus^{n+i+1} a) \mathcal{R} (Q_j \setminus^{n+j} a)$. □

Proposition A8 *Suppose* $\alpha = \mathtt{rd}\, a$ *or* $\alpha = \mathtt{wt}\, a$. *If* $M \backsimeq N$ *then* $\alpha.M \backsimeq \alpha.N$.

Proof. Consider the case $\alpha = \mathtt{rd}\, a$. It suffices to show that $\mathcal{R} \subseteq \backsimeq$, where

$$\begin{aligned}\mathcal{R} = \{&((\mathtt{rd}\, a.M)_{\sigma;\varepsilon}, (\mathtt{rd}\, a.N)_{\sigma;\varepsilon}), ((\mathtt{retry})_{\sigma;\varepsilon}, (\mathtt{retry})_{\sigma;\varepsilon})\} \cup \\ &\{((M')_{\sigma;\mathtt{rd}\, a.\delta}, (N')_{\sigma;\mathtt{rd}\, a.\delta'}) \mid ((M')_{\sigma\setminus\{a\};\delta}, (N')_{\sigma\setminus\{a\};\delta'}) \in \backsimeq, \\ &\quad (M)_{\sigma\setminus\{a\};\varepsilon} \Rightarrow (M')_{\sigma\setminus\{a\};\delta} \text{ and } (N)_{\sigma\setminus\{a\};\varepsilon} \Rightarrow (N')_{\sigma\setminus\{a\};\delta'}\}.\end{aligned}$$

Note that $M \backsimeq N$ implies $\delta =_{\sigma\setminus\{a\}} \delta'$, thus $\mathtt{rd}\, a.\delta =_\sigma \mathtt{rd}\, a.\delta'$. □

Proposition A9 *If* $M_1 \backsimeq N_1$ *and* $M_2 \backsimeq N_2$ *then* $M_1 \,\mathtt{orElse}\, M_2 \backsimeq N_1 \,\mathtt{orElse}\, N_2$.

Proof. It suffices to show that $\mathcal{R} \subseteq \backsimeq$, where

$$\begin{aligned}\mathcal{R} = \{&((M_1 \,\mathtt{orElse}\, M_2)_{\sigma;\varepsilon}, (N_1 \,\mathtt{orElse}\, N_2)_{\sigma;\varepsilon})\} \\ \cup\ \{&((A \,\mathtt{orElse}\, B), (C \,\mathtt{orElse}\, D)) \big| (M_1)_{\sigma;\varepsilon} \Rightarrow A, (M_2)_{\sigma;\varepsilon} \Rightarrow B, (N_1)_{\sigma;\varepsilon} \Rightarrow C, \\ &\qquad (N_2)_{\sigma;\varepsilon} \Rightarrow D, (A,C) \in \backsimeq, (B,D) \in \backsimeq\} \\ \cup\ \{&(B,D) \big| (M_1)_{\sigma';\varepsilon} \Rightarrow (\mathtt{retry})_{\sigma';\delta}, (N_1)_{\sigma';\varepsilon} \Rightarrow (\mathtt{retry})_{\sigma';\delta'}, (M_2)_{\sigma';\varepsilon} \Rightarrow B, \\ &\qquad (N_2)_{\sigma';\varepsilon} \Rightarrow D, (B,D) \in \backsimeq\} \\ \cup\ \{&((\mathtt{end})_{\sigma'';\delta}, (\mathtt{end})_{\sigma'';\delta'}) \,\big|\, (M_1)_{\sigma'';\varepsilon} \Rightarrow (\mathtt{end})_{\sigma'';\delta}, (N_1)_{\sigma'';\varepsilon} \Rightarrow (\mathtt{end})_{\sigma'';\delta'}\}.\end{aligned}$$

Note that $M_i \backsimeq N_i$, for $i = 1, 2$, ensures that, in case of successful termination, the resulting logs have the same effects. □

We can now prove the main results of Section 4.

Theorem A1 (Theorem 1) *Weak asynchronous bisimulation $\approx_a$ is a congruence.*

Proof. The result follows by Propositions A4-A7. □

Theorem A2 (Theorem 2) *Weak atomic equivalence $\backsimeq$ is a congruence such that $M \backsimeq N$ implies that for every σ and δ $\{\!|(M)_{\sigma;\delta}|\!\}_M \approx_a \{\!|(N)_{\sigma;\delta}|\!\}_N$ and $\mathtt{atom}(M) \approx_a \mathtt{atom}(N)$.*

Proof. By Propositions A8 and A9 we obtain that $\backsimeq$ is a congruence.

We prove that $M \backsimeq N$ implies $\{\!|(M)_{\sigma;\delta}|\!\}_M \approx_a \{\!|(N)_{\sigma;\delta}|\!\}_N$ for every σ and δ by defining the following relation and proving that it is a weak asynchronous bisimulation:

$$\mathcal{R} = \big\{ (\mathtt{atom}(M), \mathtt{atom}(N)),\ (\{\!|(M')_{\sigma;\delta\delta_M}|\!\}_M, \{\!|(N')_{\sigma;\delta\delta_N}|\!\}_N) \\ \mid (M)_{\sigma\setminus \mathrm{rd}(\delta);\varepsilon} \Rightarrow (M')_{\sigma\setminus \mathrm{rd}(\delta);\delta_M},\ (N)_{\sigma\setminus \mathrm{rd}(\delta);\varepsilon} \Rightarrow (N')_{\sigma\setminus \mathrm{rd}(\delta);\delta_N},\ M' \backsimeq N' \big\}\ .$$

As usual, the proof proceeds by a straightforward induction on the transitions the left terms (resp. the right ones) can perform.

We prove that $M \backsimeq N$ implies $\mathtt{atom}(M) \approx_a \mathtt{atom}(N)$ by contradiction. Suppose that $\mathtt{atom}(M) \not\approx_a \mathtt{atom}(N)$. This means that there is a δ such that $\mathtt{atom}(M) \stackrel{\mathrm{rd}(\delta)}{\Longrightarrow} P$, with $P = \prod_{b\in \mathrm{wt}(\delta)} \overline{b}$, and for every δ' such that $\mathtt{atom}(N) \stackrel{\mathrm{rd}(\delta')}{\Longrightarrow} Q$, with $Q = \prod_{b\in \mathrm{wt}(\delta')} \overline{b}$, we have $(P \mid \prod_{b\in(\mathrm{rd}(\delta')\setminus\mathrm{rd}(\delta))} \overline{b}) \not\approx_a (Q \mid \prod_{b\in(\mathrm{rd}(\delta)\setminus\mathrm{rd}(\delta'))} \overline{b})$. This means that there is an a such that $(P \mid \prod_{b\in(\mathrm{rd}(\delta')\setminus\mathrm{rd}(\delta))} \overline{b}) \stackrel{\overline{a}}{\longrightarrow}$ and $(Q \mid \prod_{b\in(\mathrm{rd}(\delta)\setminus\mathrm{rd}(\delta'))} \overline{b}) \not\stackrel{\overline{a}}{\longrightarrow}$ (or vice versa).

By rules (atPass) and (atOk) and definition of $\stackrel{\mu}{\longrightarrow}$, $\mathtt{atom}(M) \stackrel{\mathrm{rd}(\delta)}{\Longrightarrow} P$ implies that there is a σ such that $(M)_{\sigma;\varepsilon} \Rightarrow (\mathtt{end})_{\sigma;\delta} \stackrel{\mathrm{rd}(\delta)}{\longrightarrow} P$. By definition of $\backsimeq$ there is a δ'' such that $(N)_{\sigma;\varepsilon} \Rightarrow (\mathtt{end})_{\sigma;\delta''}$, with $\delta =_\sigma \delta''$, that is $\sigma \setminus \mathrm{rd}(\delta) \uplus \mathrm{wt}(\delta) = \sigma \setminus \mathrm{rd}(\delta'') \uplus \mathrm{wt}(\delta'')$. Thus by rules (atPass) and (atOk) and Proposition 1 $\mathtt{atom}(N) \stackrel{\mathrm{rd}(\delta'')}{\Longrightarrow} Q$ with $Q = \prod_{b\in \mathrm{wt}(\delta'')} \overline{b}$.

Suppose $P = \prod_{b\in \mathrm{wt}(\delta)} \overline{b} \stackrel{\overline{a}}{\longrightarrow}$; this means that $a \in \mathrm{wt}(\delta)$. From $\sigma \setminus \mathrm{rd}(\delta) \uplus \mathrm{wt}(\delta) = \sigma \setminus \mathrm{rd}(\delta'') \uplus \mathrm{wt}(\delta'')$ we obtain $\mathrm{wt}(\delta) = \mathrm{wt}(\delta'') \uplus \mathrm{rd}(\delta) \setminus \mathrm{rd}(\delta'')$, hence or $Q = \prod_{b\in \mathrm{wt}(\delta'')} \overline{b} \stackrel{\overline{a}}{\longrightarrow}$ or $\prod_{b\in(\mathrm{rd}(\delta)\setminus\mathrm{rd}(\delta''))} \overline{b} \stackrel{\overline{a}}{\longrightarrow}$.

Suppose $a \in (\mathrm{rd}(\delta'') \setminus \mathrm{rd}(\delta))$, then $\mathrm{wt}(\delta'') = \mathrm{wt}(\delta) \uplus \mathrm{rd}(\delta'') \setminus \mathrm{rd}(\delta)$ implies that $a \in \mathrm{wt}(\delta'')$, that is $Q \stackrel{\overline{a}}{\longrightarrow}$.

In both cases we have a contradiction because we have assumed that $(Q \mid \prod_{b\in(\mathrm{rd}(\delta)\setminus\mathrm{rd}(\delta''))} \overline{b}) \not\stackrel{\overline{a}}{\longrightarrow}$. □

Note that weak atomic bisimulation entails weak asynchronous bisimulation, but the inverse does not hold. E.g. $\mathtt{atom}(\mathtt{rd}\,a\,.\mathtt{wt}\,a\,.\mathtt{end}) \approx_a \mathtt{atom}(\mathtt{end})$ but $\mathtt{rd}\,a\,.\mathtt{wt}\,a\,.\mathtt{end} \not\approx \mathtt{end}$.

B Proofs of laws in Table 4

Laws in Table 4 are proved, as usual, by showing appropriate bisimulation relations. In the following cases $\mathcal{R}$ is the proposed bisimulation. In what follows $a \notin \sigma$ means that the name a does not appear in σ and $a^n \in \sigma$ means that σ contains n copies of a.

(comm) $\alpha.\alpha'.M \backsimeq \alpha'.\alpha.M$**:** Suppose $\alpha = \mathtt{rd}\,a$ and $\alpha' = \mathtt{rd}\,b$ (the other cases are similar.)

$$\begin{aligned}
\mathcal{R} = {} & \left\{((\mathtt{rd}\,a.\mathtt{rd}\,b.M)_{\sigma;\varepsilon}, (\mathtt{rd}\,b.\mathtt{rd}\,a.M)_{\sigma;\varepsilon})\right\} \\
& \cup \{((\mathtt{rd}\,b.M)_{\sigma;\mathtt{rd}\,a}, (\mathtt{rd}\,a.M)_{\sigma;\mathtt{rd}\,b}),\ ((M'')_{\sigma;\mathtt{rd}\,a.\mathtt{rd}\,b.\delta}, (M'')_{\sigma;\mathtt{rd}\,b.\mathtt{rd}\,a.\delta}) \\
& \qquad \left| a^n, b^m \in \sigma,\ n,m > 0,\ (M)_{\sigma\setminus\{a,b\};\varepsilon} \Rightarrow (M'')_{\sigma\setminus\{a,b\};\delta}\right\} \\
& \cup \left\{((\mathtt{retry})_{\sigma;\varepsilon}, (\mathtt{retry})_{\sigma;\varepsilon}) \middle| a, b \notin \sigma\right\} \\
& \cup \left\{((\mathtt{retry})_{\sigma;\varepsilon}, (\mathtt{rd}\,a.M)_{\sigma;\mathtt{rd}\,b}),\ ((\mathtt{retry})_{\sigma;\varepsilon}, (\mathtt{retry})_{\sigma;\mathtt{rd}\,b}),\right. \\
& \qquad \left. \left| a \notin \sigma,\ b^m \in \sigma,\ m > 0\right.\right\} \\
& \cup \left\{((\mathtt{rd}\,b.M)_{\sigma;\mathtt{rd}\,a}, (\mathtt{retry})_{\sigma;\varepsilon}),\ ((\mathtt{retry})_{\sigma;\mathtt{rd}\,a}, (\mathtt{retry})_{\sigma;\varepsilon})\right. \\
& \qquad \left. \left| a^n \in \sigma,\ b \notin \sigma,\ n > 0\right.\right\}.
\end{aligned}$$

(dist) $\alpha.(M\ \mathtt{orElse}\ N) \backsimeq (\alpha.M)\ \mathtt{orElse}\ (\alpha.N)$**:** Suppose $M' = \mathtt{rd}\,a.(M\ \mathtt{orElse}\ N)$ and $N' = (\mathtt{rd}\,a.M)\ \mathtt{orElse}\ (\mathtt{rd}\,a.N)$.

$$\begin{aligned}
\mathcal{R} = {} & \left\{((M')_{\sigma;\varepsilon}, (N')_{\sigma;\varepsilon}),\ ((M')_{\sigma;\varepsilon}, (\mathtt{rd}\,a.M)_{\sigma;\varepsilon}\ \mathtt{orElse}\ (\mathtt{rd}\,a.N)_{\sigma;\varepsilon})\right\} \\
& \cup \left\{((\mathtt{retry})_{\sigma;\varepsilon}, (\mathtt{retry})_{\sigma;\varepsilon}\ \mathtt{orElse}\ (\mathtt{rd}\,a.N)_{\sigma;\varepsilon}),\right. \\
& \quad ((\mathtt{retry})_{\sigma;\varepsilon}, (\mathtt{rd}\,a.M)_{\sigma;\varepsilon}\ \mathtt{orElse}\ (\mathtt{retry})_{\sigma;\varepsilon}), \\
& \quad ((\mathtt{retry})_{\sigma;\varepsilon}, (\mathtt{retry})_{\sigma;\varepsilon}\ \mathtt{orElse}\ (\mathtt{retry})_{\sigma;\varepsilon}),\ ((\mathtt{retry})_{\sigma;\varepsilon}, (\mathtt{retry})_{\sigma;\varepsilon}) \\
& \qquad \left. \left| a \notin \sigma\right.\right\} \\
& \cup \left\{((M\ \mathtt{orElse}\ N)_{\sigma;\mathtt{rd}\,a}, (M)_{\sigma;\mathtt{rd}\,a}\ \mathtt{orElse}\ (N)_{\sigma;\mathtt{rd}\,a}),\right. \\
& \quad ((M')_{\sigma;\varepsilon}, (M)_{\sigma;\mathtt{rd}\,a}\ \mathtt{orElse}\ (\mathtt{rd}\,a.N)_{\sigma;\varepsilon}),\ ((M')_{\sigma;\varepsilon}, (\mathtt{rd}\,a.M)_{\sigma;\varepsilon}\ \mathtt{orElse}\ (N)_{\sigma;\mathtt{rd}\,a}) \\
& \qquad \left. \left| a^n \in \sigma,\ n > 0\right.\right\} \\
& \cup \left\{(A\ \mathtt{orElse}\ (N)_{\sigma;\mathtt{rd}\,a}, A\ \mathtt{orElse}\ (\mathtt{rd}\,a.N)_{\sigma;\varepsilon}) \,|\, (M)_{\sigma;\mathtt{rd}\,a} \Rightarrow A,\ a^n \in \sigma,\ n > 0\right\} \\
& \cup \left\{((M)_{\sigma;\mathtt{rd}\,a}\ \mathtt{orElse}\ B, (\mathtt{rd}\,a.M)_{\sigma;\varepsilon}\ \mathtt{orElse}\ B) \,|\, (N)_{\sigma;\mathtt{rd}\,a} \Rightarrow B,\ a^n \in \sigma,\ n > 0\right\} \\
& \cup \left\{(A\ \mathtt{orElse}\ B, A\ \mathtt{orElse}\ B) \,|\, (M)_{\sigma;\mathtt{rd}\,a} \Rightarrow A,\ (N)_{\sigma;\mathtt{rd}\,a} \Rightarrow B,\ a^n \in \sigma,\ n > 0\right\} \\
& \cup \left\{(C, C) \,|\, (M)_{\sigma;\mathtt{rd}\,a} \Rightarrow (\mathtt{retry})_{\sigma;\delta},\ (N)_{\sigma;\mathtt{rd}\,a} \Rightarrow C,\ a^n \in \sigma,\ n > 0\right\}.
\end{aligned}$$

(ass) M_1 $\texttt{orElse}$ $(M_2$ $\texttt{orElse}$ $M_3) \backsimeq (M_1$ $\texttt{orElse}$ $M_2)$ $\texttt{orElse}$ M_3**:**

$$\begin{aligned}
\mathcal{R} = &\big\{((M_1 \texttt{ orElse } (M_2 \texttt{ orElse } M_3))_{\sigma;\varepsilon}, ((M_1 \texttt{ orElse } M_2) \texttt{ orElse } M_3)_{\sigma;\varepsilon}),\\
&\quad ((M_1)_{\sigma;\varepsilon} \texttt{ orElse } (M_2 \texttt{ orElse } M_3)_{\sigma;\varepsilon}, (M_1 \texttt{ orElse } M_2)_{\sigma;\varepsilon} \texttt{ orElse } (M_3)_{\sigma;\varepsilon})\big\}\\
&\cup \big\{(A \texttt{ orElse } (B \texttt{ orElse } C), (A \texttt{ orElse } B) \texttt{ orElse } C),\\
&\quad (A \texttt{ orElse } (M_2 \texttt{ orElse } M_3)_{\sigma;\varepsilon}, (A \texttt{ orElse } (M_2)_{\sigma;\varepsilon}) \texttt{ orElse } (M_3)_{\sigma;\varepsilon}),\\
&\quad ((M_1)_{\sigma;\varepsilon} \texttt{ orElse } ((M_2)_{\sigma;\varepsilon} \texttt{ orElse } C), (M_1 \texttt{ orElse } M_2)_{\sigma;\varepsilon} \texttt{ orElse } C),\\
&\quad ((M_1)_{\sigma;\varepsilon} \texttt{ orElse } (B \texttt{ orElse } (M_3)_{\sigma;\varepsilon}), ((M_1)_{\sigma;\varepsilon} \texttt{ orElse } B) \texttt{ orElse } (M_3)_{\sigma;\varepsilon})\\
&\qquad \big|(M_1)_{\sigma;\varepsilon} \Rightarrow A, (M_2)_{\sigma;\varepsilon} \Rightarrow B, (M_3)_{\sigma;\varepsilon} \Rightarrow C\big\}\\
&\cup \big\{((M_2 \texttt{ orElse } M_3)_{\sigma';\varepsilon}, (M_2)_{\sigma';\varepsilon} \texttt{ orElse } (M_3)_{\sigma';\varepsilon}), ((D \texttt{ orElse } E), D \texttt{ orElse } E)\\
&\qquad \big|(M_1)_{\sigma';\varepsilon} \Rightarrow (\texttt{retry})_{\sigma';\delta}, (M_2)_{\sigma';\varepsilon} \Rightarrow D, (M_3)_{\sigma';\varepsilon} \Rightarrow E\big\}\\
&\cup \big\{(F,F)\big|(M_1)_{\sigma'';\varepsilon} \Rightarrow (\texttt{retry})_{\sigma';\delta}, (M_2)_{\sigma'';\varepsilon} \Rightarrow (\texttt{retry})_{\sigma'';\delta'}, (M_3)_{\sigma'';\varepsilon} \Rightarrow F\big\}\ .
\end{aligned}$$

(absRt1) $\alpha.\texttt{retry} \backsimeq \texttt{retry}$**:** suppose $\alpha = \texttt{rd}\,a$:

$$\begin{aligned}
\mathcal{R} = &\big\{((\texttt{rd}\,a.\texttt{retry})_{\sigma;\varepsilon}, (\texttt{retry})_{\sigma;\varepsilon})\big\}\\
&\cup \big\{((\texttt{retry})_{\sigma;\texttt{rd}\,a}, (\texttt{retry})_{\sigma;\varepsilon})\big|a^n \in \sigma, n > 0\big\}\\
&\cup \big\{((\texttt{retry})_{\sigma;\varepsilon}, (\texttt{retry})_{\sigma;\varepsilon})|a \notin \sigma\big\}\ .
\end{aligned}$$

(absRt2) $\texttt{retry}$ $\texttt{orElse}$ $M \backsimeq M \backsimeq M$ $\texttt{orElse}$ $\texttt{retry}$**:**

$$\begin{aligned}
\mathcal{R}_1 = &\big\{((\texttt{retry } \texttt{orElse } M)_{\sigma;\varepsilon}, (M)_{\sigma;\varepsilon})\big\}\\
&\cup \big\{((\texttt{retry})_{\sigma;\varepsilon} \texttt{ orElse } A, A), (A,A)\big|(M)_{\sigma;\varepsilon} \Rightarrow A\big\}\\
\mathcal{R}_2 = &\big\{((M \texttt{ orElse } \texttt{retry})_{\sigma;\varepsilon}, (M)_{\sigma;\varepsilon})\big\}\\
&\cup \big\{(A \texttt{ orElse } (\texttt{retry})_{\sigma;\varepsilon}, A)\big|(M)_{\sigma;\varepsilon} \Rightarrow A\big\}\\
&\cup \big\{((\texttt{end})_{\sigma;\delta}, (\texttt{end})_{\sigma;\delta})\big|(M)_{\sigma;\varepsilon} \Rightarrow (\texttt{end})_{\sigma;\delta}\big\}\\
&\cup \big\{((\texttt{retry})_{\sigma;\varepsilon}, (\texttt{retry})_{\sigma;\delta})\big|(M)_{\sigma;\varepsilon} \Rightarrow (\texttt{retry})_{\sigma;\delta}\big\}\ .
\end{aligned}$$

(absEnd) $\texttt{end}$ $\texttt{orElse}$ $M \backsimeq \texttt{end}$**:**

$$\begin{aligned}
\mathcal{R} = &\big\{((\texttt{end } \texttt{orElse } M)_{\sigma;\varepsilon}, (\texttt{end})_{\sigma;\varepsilon}), ((\texttt{end})_{\sigma;\varepsilon}, (\texttt{end})_{\sigma;\varepsilon})\big\}\\
&\cup \big\{((\texttt{end})_{\sigma;\varepsilon} \texttt{ orElse } A, (\texttt{end})_{\sigma;\varepsilon})\ \big|(M)_{\sigma;\varepsilon} \Rightarrow A\big\}\ .
\end{aligned}$$

(asy) $a.\overline{a} \approx_a \mathbf{0}$**:**

$$\mathcal{R} = \big\{(a.\overline{a}, \mathbf{0}), (\overline{a}, \overline{a}), (\mathbf{0}, \mathbf{0})\big\}\ .$$

(a-asy) $\texttt{atom}(\texttt{rd}\,a.\texttt{wt}\,a.\texttt{end}) \approx_a \mathbf{0}$**:**

$$\begin{aligned}
\mathcal{R} = &\big\{(\texttt{atom}(\texttt{rd}\,a.\texttt{wt}\,a.\texttt{end}), \mathbf{0}), (\{\!|(\texttt{rd}\,a.\texttt{wt}\,a.\texttt{end})_{\sigma;\varepsilon}|\!\}_{\texttt{rd}\,a.\texttt{wt}\,a.\texttt{end}}, \mathbf{0})\big\}\\
&\cup \big\{(\{\!|(\texttt{wt}\,a.\texttt{end})_{\sigma;\texttt{rd}\,a}|\!\}_{\texttt{rd}\,a.\texttt{wt}\,a.\texttt{end}}, \mathbf{0}), (\{\!|(\texttt{end})_{\sigma;\texttt{rd}\,a.\texttt{wt}\,a}|\!\}_{\texttt{rd}\,a.\texttt{wt}\,a.\texttt{end}}, \mathbf{0})\big|a^n \in \sigma, n > 0\big\}\\
&\cup \big\{(\{\!|(\texttt{retry})_{\sigma;\varepsilon}|\!\}_{\texttt{rd}\,a.\texttt{wt}\,a.\texttt{end}}, \mathbf{0}), (\overline{a}, \overline{a}), (\mathbf{0}, \mathbf{0})\big|a \notin \sigma\big\}\ .
\end{aligned}$$

(a-1) $\mathtt{atom}(\mathtt{rd}\,a\,.\mathtt{end}) \approx_a a$**:**

$$\begin{aligned}\mathcal{R} = &\left\{(\mathtt{atom}(\mathtt{rd}\,a\,.\mathtt{end}), a),\ (\{\!|(\mathtt{rd}\,a\,.\mathtt{end})_{\sigma;\varepsilon}|\!\}_{\mathtt{rd}\,a\,.\mathtt{end}}, a)\right\}\\ &\cup \left\{(\{\!|(\mathtt{end})_{\sigma;\mathtt{rd}\,a}|\!\}_{\mathtt{rd}\,a\,.\mathtt{end}}, a),\ (\mathbf{0},\mathbf{0}) \middle| a^n \in \sigma, n>0\right\}\\ &\cup \left\{(\{\!|(\mathtt{retry})_{\sigma;\varepsilon}|\!\}_{\mathtt{rd}\,a\,.\mathtt{end}}, a) \middle| a \notin \sigma\right\}\ .\end{aligned}$$

C Proof of Prposition 2

In this section we show that laws in Table 4 can be used for eliminating redundant branches from an atomic expression and obtaining an equivalent expression in normal form (see proof of Proposition 2.) Some preliminary results are needed.

The next proposition states that if K''s reads include K's then K' is bigger than K in our weak atomic preorder.

Proposition C1 *Suppose* $K = A_1.\cdots.A_n$ *and* $K' = B_1.\cdots.B_m$*, with* $A_i, B_j ::= \mathtt{rd}\,a \,|\, \mathtt{wt}\,a$*. If* $\mathrm{rd}(K) \subseteq \mathrm{rd}(K')$ *then* $K \sqsupseteq K'$*.*

Proof. It is enough to observe that if $(K')_{\sigma;\varepsilon} \Rightarrow (\mathtt{end})_{\sigma;\delta}$ then $\mathrm{rd}(K') \subseteq \sigma$ (rules (ARdOk) and (ARdF)); thus $\mathrm{rd}(K) \subseteq \sigma$, and by (ARdOk) we get $(K)_{\sigma;\varepsilon} \Rightarrow (\mathtt{end})_{\sigma;\delta'}$. □

As a consequence of the previous proposition, we obtain that, in an `orElse` expression, a redundant branch, that is a branch which includes the reads of at least one of its preceding branches, can be eliminated.

Proposition C2 *Consider the expressions* $K_1, \ldots, K_n$ *where, for* $i = 1, \ldots, n$*,* K_i *is of the form* $A_{i_1}.\cdots.A_{i_{n_i}}$ *with* $A_{i_j} ::= \mathtt{rd}\,a \,|\, \mathtt{wt}\,a$*. If* $\mathrm{rd}(K_j) \subseteq \mathrm{rd}(K_n)$*, for a* j *such that* $0 < j < n$*, then*

$$K_1\ \mathtt{orElse}\ \cdots\ \mathtt{orElse}\ K_{n-1}\ \mathtt{orElse}\ K_n \backsimeq K_1\ \mathtt{orElse}\ \cdots\ \mathtt{orElse}\ K_{n-1}\ .$$

Proof. The proof proceeds by using Proposition C1, the fact that $M \sqcup M' \backsimeq M$ if and only if $M \sqsupseteq M'$ (see pag. 13) and `orElse` 's rules in Table 3. □

As previously said, the proof of the following theorem show how to apply rules in Table 4 for rearranging an atomic expression into an equivalent one in normal form.

Proposition C3 (Proposition 2) *For every expression* M *there is an expression* M' *in normal form such that* $M \backsimeq M'$*.*

Proof. The proof proceeds by induction on the structure of M:

$M = \mathtt{end}$**:** $M' = M = \mathtt{end}$;
$M = \mathtt{retry}$**:** $M' = M = \mathtt{retry}$;
$M = \alpha.N$**:** by induction hypothesis, there is an N' in normal form such that $N \backsimeq N'$. By Proposition A8, $\alpha.N \backsimeq \alpha.N'$, thus by choosing $M' = \alpha.N'$ we obtain $M \backsimeq M'$;
$M = N\ \mathtt{orElse}\ N'$**:** by induction hypothesis, there are N_0 and N_0', in normal form, such that $N \backsimeq N_0$ and $N' \backsimeq N_0'$. By Proposition A9, $M = N\ \mathtt{orElse}\ N' \backsimeq N_0\ \mathtt{orElse}\ N_0'$. We choose M' by considering the following cases:
- if $N_0 = \mathtt{retry}$ we choose $M' = N_0'$, because, by (absRt), $\mathtt{retry}\ \mathtt{orElse}\ N_0' \backsimeq N_0'$;

– if $N_0 = N_{0_1}$ orElse … orElse N_{0_n} and $N'_0 = N'_{0_1}$ orElse ⋯ orElse N'_{0_m}, consider $P = \{j \mid k \in \{1,\ldots,n\} :\ \mathrm{rd}(N_{0_k}) \subseteq \mathrm{rd}(N'_{0_j})\}$. If $P = \emptyset$ this means that $M' = N_0$ orElse N'_0 is in normal form.
Otherwise, suppose $P = \{j_1,\ldots,j_l\}$ with $j_i < j_w$ for $i < w$; by applying Proposition C2 and A9 and (ass) at every step, we have

$$
\begin{aligned}
& N_0 \texttt{ orElse } N'_0 \\
\preceq\ & \quad \text{(by removing } N'_{0_{j_1}}\text{)} \\
& N_0 \texttt{ orElse } N'_{0_1} \texttt{ orElse } \cdots \texttt{ orElse } N'_{0_{j_1-1}} \texttt{ orElse } N'_{0_{j_1+1}} \texttt{ orElse } \cdots \texttt{ orElse } N'_{0_m} \\
\preceq\ & \quad \text{(by removing } N'_{0_{j_2}}\text{)} \\
& N_0 \texttt{ orElse } N'_{0_1} \texttt{ orElse } \cdots \texttt{ orElse } N'_{0_{j_1-1}} \texttt{ orElse } N'_{0_{j_1+1}} \texttt{ orElse } \cdots \\
& \qquad \texttt{orElse } N'_{0_{j_2-1}} \texttt{ orElse } N'_{0_{j_2+1}} \texttt{ orElse } \cdots \texttt{ orElse } N'_{0_m} \\
\preceq\ & \quad \text{(by removing } N'_{0_{j_3}}\text{)} \\
& \vdots \\
\preceq\ & \quad \text{(by removing } N'_{0_{j_l}}\text{)} \\
& N_0 \texttt{ orElse } N'_{0_1} \texttt{ orElse } \cdots \texttt{ orElse } N'_{0_{j_1-1}} \texttt{ orElse } N'_{0_{j_1+1}} \texttt{ orElse } \cdots \\
& \qquad \texttt{orElse } N'_{0_{j_2-1}} \texttt{ orElse } N'_{0_{j_2+1}} \texttt{ orElse } \cdots \texttt{ orElse } N'_{0_{j_l-1}} \\
& \qquad \texttt{orElse } N'_{0_{j_l+1}} \texttt{ orElse } \cdots \texttt{ orElse } N'_{0_m} \\
=\ & M' \text{ (that is in normal form.)}
\end{aligned}
$$

In every case, $M' \preceq N_0$ orElse N'_0, thus $M \preceq M'$. □

D Proofs of Section 5

Lemma D1 (Lemma 1) *Assume that $s' \preceq s$ and $P \overset{\bar{s}}{\Rightarrow} P'$, then there is a process P'' such that $P \overset{\bar{s}'}{\Rightarrow} P''$.*

Proof. $s' \preceq s$ means $s' \preceq_0^n s$, for some $n \geq 0$. The proof proceeds by induction on n. For $n = 0$ we have $s = s'$. Suppose $n > 0$ and $s' \preceq_0^{n-1} s'' \preceq_0 s$. The result follows by induction hypothesis if we show that $P \overset{\bar{s}''}{\Rightarrow}$. We proceed by distinguishing the possible cases for $s'' \preceq_0 s$ according to laws (TO1)-(TO4).

(TO1) $s'' = rr'$ and $s = r\{a\}r'$, thus $\overline{s''} = \bar{r}\overline{r'}$ and $\bar{s} = \bar{r}\bar{a}\overline{r'}$. $P \overset{\bar{s}}{\Rightarrow}$ implies $P \overset{\bar{r}}{\Rightarrow} P_1 \overset{\bar{a}}{\Rightarrow} P_2 \overset{\overline{r'}}{\Rightarrow}$, and by Proposition A1, $P_1 \sim P_2 \,|\, \bar{a}$, that is $P \overset{\bar{r}}{\Rightarrow} P_2 \,|\, \bar{a} \overset{\overline{r'}}{\Rightarrow}$, hence $P \overset{\overline{s''}}{\Rightarrow}$;

(TO2) $s'' = rl\{a\}r'$ and $s = r\{a\}lr'$, thus $\overline{s''} = \bar{r}\bar{l}\bar{a}\overline{r'}$ and $\bar{s} = \bar{r}\bar{a}\overline{lr'}$. $P \overset{\bar{s}}{\Rightarrow}$ implies $P \overset{\bar{r}}{\Rightarrow} P_1 \overset{\bar{a}}{\Rightarrow} P_2 \overset{\bar{l}}{\Rightarrow} P_3 \overset{\overline{r'}}{\Rightarrow}$, and by Proposition A1, $P_1 \sim P_2 \,|\, \bar{a}$, that is $P \overset{\bar{r}}{\Rightarrow} P_2 \,|\, \bar{a} \overset{\bar{l}}{\Rightarrow} P_3 \,|\, \bar{a} \overset{\overline{r'}}{\Rightarrow}$, hence $P \overset{\overline{s''}}{\Rightarrow}$;

(TO3) $s'' = rr'$ and $s = r\{a\}\bar{a}r'$, thus $\overline{s''} = \bar{r}\overline{r'}$ and $\bar{s} = \bar{r}\bar{a}\{a\}\overline{r'}$. $P \overset{\bar{s}}{\Rightarrow}$ implies $P \overset{\bar{r}}{\Rightarrow} P_1 \overset{\bar{a}}{\Rightarrow} P_2 \overset{a}{\Rightarrow} P_3 \overset{\overline{r'}}{\Rightarrow}$, hence, by Proposition A1, $P_1 \sim P_2 \,|\, \bar{a}$, that is P_2 can synchronize with $\bar{a}$ and $P \overset{\bar{r}}{\Rightarrow} P_2 \,|\, \bar{a} \Rightarrow P_3 \overset{\overline{r'}}{\Rightarrow}$, that is $P \overset{\overline{s''}}{\Rightarrow}$;

(TO4) $s'' = \{a_1\}\cdots\{a_n\}$ and $s = \{a_1,\cdots,a_n\}$, or viceversa; in this case $\overline{s} = \overline{s''}$ by definition of $\bar{\cdot}$. □

Lemma D2 (Lemma 2) *Consider two traces s and r. If there is a process Q such that $O(s)\stackrel{\overline{r}}{\Rightarrow}\stackrel{\overline{w}}{\Rightarrow}Q$ then $r \preceq s$.*

Proof. The proof proceeds by induction on s.

$s = \overline{a}s'$**:** $O(s) = a.O(s')$ and $O(s)\stackrel{\overline{rw}}{\Rightarrow}$ implies $\overline{r} = a\overline{r'}$ such that $O(s)\xrightarrow{\{a\}}O(s')\stackrel{\overline{r'}}{\Rightarrow}$. By induction hypothesis, $r' \preceq s'$, hence by prefixing, $r = \overline{a}r' \preceq \overline{a}s' = s$;

$s = \{a_1,\cdots,a_n\}s'$**:** $O(s) = \left(\prod_{a\in\{a_1,\cdots,a_n\}}\overline{a}\right) \mid O(s')$. We have $O(s)\stackrel{\overline{rw}}{\Rightarrow}$, we can distinguish the following cases depending on $\overline{r}$:

$\overline{a_i} \notin \overline{r}$**:** by induction hypothesis, $O(s')\stackrel{\overline{rw}}{\Rightarrow}$ implies $r \preceq s'$ and by (TO1), $r \preceq s' \preceq \{a_1\}\cdots\{a_n\}s' \;_0\!\succeq\!\preceq_0 \{a_1,\cdots,a_n\}s' = s$;

$\overline{a_{i_1}},\cdots\overline{a_{i_k}} \in \overline{r}$ **for** $\{a_{i_1},\cdots,a_{i_k}\} \subseteq \{a_1,\cdots,a_n\}$**:** in this case $\overline{r} = \overline{r_1 a_{i_1}}\cdots\overline{r_k a_{i_k} r_{k+1}}$ and $O(s')\xrightarrow{\overline{r_1}\cdots\overline{r_{k+1}}\overline{w}}$. By induction hypothesis, $r_1\cdots r_{k+1} \preceq s'$:

$$\begin{array}{rcll} r & = & r_1\{a_{i_1}\}\cdots r_k\{a_{i_k}\}r_{k+1} & \\ & \preceq & \{a_{i_1}\}\cdots\{a_{i_k}\}r_1\cdots r_{k+1} & \text{(by (TO2))} \\ & \preceq & \{a_{i_1}\}\cdots\{a_{i_k}\}s' & \text{(by induction and prefixing)} \\ & \preceq & \{a_1\}\cdots\{a_n\}s' & \text{(by (TO1) and (TO2))} \\ & {}_0\!\succeq\!\preceq_0 & \{a_1\cdots a_n\}s' & \text{(by (TO4))} \\ & = & s; & \end{array}$$

$\overline{r} = \overline{r_1}\cdots\overline{r_k}$ **and** $O(s')\xrightarrow{\overline{r_1}\{a_{i_1}\}\cdots\overline{r_k}\{a_{i_k}\}r_{k+1}}$ **for** $\{a_{i_1},\cdots,a_{i_k}\} \subseteq \{a_1,\cdots,a_n\}$**:** by induction hypothesis, $r_1\overline{a_{i_1}}\cdots r_k\overline{a_{i_k}}r_{k+1} \preceq s'$ and:

$$\begin{array}{rcll} r & = & r_1\cdots r_k & \\ & \preceq & r_1\{a_{i_1}\}\overline{a_{i_1}}\cdots r_k\{a_{i_k}\}\overline{a_{i_k}}r_{k+1} & \text{(by (TO3))} \\ & \preceq & \{a_{i_1}\}\cdots\{a_{i_k}\}r_1\overline{a_{i_1}}\cdots r_k\overline{a_{i_k}}r_{k+1} & \text{(by (TO2))} \\ & \preceq & \{a_{i_1}\}\cdots\{a_{i_k}\}s' & \text{(by induction)} \\ & \preceq & \{a_1\}\cdots\{a_n\}s_0 & \text{(by (TO1) and (TO2))} \\ & {}_0\!\succeq\!\preceq_0 & \{a_1\cdots a_n\}s' & \text{(by (TO4))} \\ & = & s. & \end{array}$$

□

The proof of the full-abstraction theorem is standard (see e.g. [7]).

Theorem D1 (Theorem 3) *For all processes P and Q, $P \sqsubseteq_{may} Q$ if and only if $P \ll_{may} Q$.*

Proof. ⇒**:** Suppose $P \ll_{may} Q$ and P *may* O for any observer O we have to show that Q *may* O. P *may* O means that $P \mid O\stackrel{\overline{w}}{\Rightarrow}$, that is there exists a trace s such that $P\stackrel{s}{\Rightarrow}$ and $O\stackrel{\overline{sw}}{\Rightarrow}$. $P \ll_{may} Q$ implies that there exists $s' \preceq s$ such that $Q\stackrel{s'}{\Rightarrow}$. $s' \preceq s$ implies $s'w \preceq sw$. By Lemma D1 and $O\stackrel{\overline{sw}}{\Rightarrow}$ we get that $O\stackrel{\overline{s'w}}{\Longrightarrow}$. Hence, from $Q\stackrel{s'}{\Rightarrow}$ we obtain $Q \mid O\stackrel{\overline{w}}{\Rightarrow}$, that is Q *may* O ($P \sqsubseteq_{may} Q$).

$\Leftarrow$**:** Suppose $P \sqsubseteq_{may} Q$ and $P \stackrel{s}{\Rightarrow}$, we have to show that there exists $s' \preceq s$ such that $Q \stackrel{s'}{\Rightarrow}$. From $P \stackrel{s}{\Rightarrow}$ and $O(s) \stackrel{\overline{sw}}{\Rightarrow}$ we have $P \mid O(s) \stackrel{\overline{w}}{\Rightarrow}$, that is P *may* $O(s)$. Hence Q *may* $O(s)$, that is $Q \mid O(s) \stackrel{\overline{w}}{\Rightarrow}$. Thus, there exists s' such that $Q \stackrel{s'}{\Rightarrow}$ and $O(s) \stackrel{s'\overline{w}}{\Longrightarrow}$, and, by Lemma D2 and $O(s) \stackrel{s'\overline{w}}{\Longrightarrow}$ we have $s' \preceq s$, that is $P \ll_{may} Q$. □

Lemma D3 (Lemma 3) *Assume* $M = \bigsqcup_{i \in 1..n} K_i$ *is an expression in normal form. For every index* i *in* $\{1,\ldots,n\}$ *we have* $\mathtt{atom}(M);\sigma_i \rightarrow^* \{\!|(\mathtt{end})_{\sigma_i;\delta}|\!\}_M;\sigma_i$ *where* $\sigma_i = \mathrm{rd}(K_i) = \mathrm{rd}(\delta)$ *and* $\mathrm{wt}(\delta) = \mathrm{wt}(K_i)$.

Proof. By definition of normal form. □

Corollary D1 *Assume* $M = \bigsqcup_{i \in 1..n} K_i$ *is an expression in normal form. The possible behavior of* $\mathtt{atom}(M)$ *can be described as* $\mathtt{atom}(M) \stackrel{\sigma_i}{\Rightarrow} \prod_{b \in \mathrm{wt}(r_i)} b$ *for every* $i \in 1..n$ *where* σ_i *is the multiset* $\mathrm{rd}(K_i)$.

Proof. By Lemma D3, rule (atOk) and definition of $\stackrel{\mu}{\rightarrow}$. □

We can prove now the main result of Section 5, that is that may-testing semantics is not able to distinguish the behavior of an atomic expression from the behavior of the corresponding CCS process.

Theorem D2 (Theorem 3) *For every expression* M *in normal form we have* $\mathtt{atom}(M) \simeq_{may} [\![M]\!]$.

Proof. The proof proceeds by using the alternative preorder instead of the may preorder; in what follows it is shown that:

1. $\mathtt{atom}(M) \ll_{may} [\![M]\!]$;
2. $[\![M]\!] \ll_{may} \mathtt{atom}(M)$.

Remember that M is in normal-form, thus $M = \mathrm{OrElse}_{i=1,\ldots,n} K_i$ and $[\![M]\!] = \sum_{i=1,\ldots,n} [\![K_i]\!]$. The two points are shown in what follows.

1. For proving that $\mathtt{atom}(M) \ll_{may} [\![M]\!]$, we have to show that $\forall s$ such that $\mathtt{atom}(M) \stackrel{s}{\Rightarrow}$ there exists $s' \preceq s$ such that $[\![M]\!] \stackrel{s'}{\Rightarrow}$. We distinguish the following cases for s:
   $s = \varepsilon$**:** in this case we can choose $s' = \varepsilon$;
   $s = \delta \overline{a}_{i_1} \cdots \overline{a}_{i_l}$ **with** $l \geq 0$**:** by Corollary D1, there is a $j \in \{1,\ldots,n\}$ such that $\delta = \mathrm{rd}(K_j)$,
   $$\mathtt{atom}(M) \stackrel{\mathrm{rd}(K_j)}{\Longrightarrow} \overline{a_1} \mid \cdots \mid \overline{a_m} \stackrel{\overline{a}_{i_1} \cdots \overline{a}_{i_l}}{\Longrightarrow}$$
   with $\{a_{i_1},\cdots,a_{i_l}\} \subseteq \{a_1,\cdots,a_m\} = \mathrm{wt}(K_j)$.
   Suppose $\mathrm{rd}(K_j) = \{b_1,\cdots,b_k\}$. By definition, $[\![K_j]\!] = b_1.\cdots.b_k.(\overline{a_1} \mid \cdots \mid \overline{a_m})$ with $\{a_1,\cdots,a_m\} = \mathrm{wt}(K_j)$. That is, if we choose the j-th summands of $[\![M]\!]$, we have $[\![M]\!] \stackrel{s'}{\Rightarrow}$ with $s' = \{b_1\}\cdots\{b_k\}\overline{a_{i_1}}\cdots\overline{a_{i_l}}$, and by (TO4) $s'_0 \succeq \preceq_0 s$;
2. For proving that $[\![M]\!] \ll_{may} \mathtt{atom}(M)$, we have to show that $\forall s$ such that $[\![M]\!] \stackrel{s}{\Rightarrow}$ there exists $s' \preceq s$ such that $\mathtt{atom}(M) \stackrel{s'}{\Rightarrow}$. We distinguish the following cases for s:
   $s = \{b_1\}\cdots\{b_k\}$**:** s contains only input actions, thus we can choose $s' = \varepsilon \preceq s$ and $\mathtt{atom}(M) \stackrel{s}{\Rightarrow}$;
   $s = \{b_1\}\cdots\{b_k\}\overline{a_1}\cdots\overline{a_m}$ **with** $m > 0$**:** in this case there is a $j \in \{1,\ldots,n\}$ such that $[\![K_j]\!] \stackrel{s}{\Rightarrow}$, $\{b_1,\cdots,b_k\} = \mathrm{rd}(K_j)$ and $\{a_1,\cdots,a_m\} \subseteq \mathrm{wt}(K_j)$ (by definition of $[\![\cdot]\!]$). Suppose $\sigma = \mathrm{rd}(K_j)$, by Lemma D3, $\mathtt{atom}(M);\sigma \Rightarrow \{\!|(\mathtt{end})_{\sigma;\delta}|\!\}_M$ with $\mathrm{rd}(\delta) = \mathrm{rd}(K_j)$ and $\mathrm{wt}(\delta) = \mathrm{wt}(K_j)$. This means that $\mathtt{atom}(M) \stackrel{\mathrm{rd}(K_j)}{\Longrightarrow} \prod_{a \in \mathrm{wt}(K_j)} \overline{a}$, that is (by (TO4)) there is an $s' = \mathrm{rd}(\delta)\overline{a_1}\cdots\overline{a_m}\,_0 \succeq \preceq_0 \{b_1\}\cdots\{b_k\}\overline{a_1}\cdots\overline{a_m} = s$ such that $\mathtt{atom}(M) \stackrel{s'}{\Rightarrow}$.

□