Axiomatising ST-Bisimulation Equivalence

Nadia Busi   Rob van Glabbeek   Roberto Gorrieri

Technical Report UBLCS-93-26

December 1993
Revised June 1994

Laboratory for Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)

The University of Bologna Laboratory for Computer Science Research Technical Reports are available via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` in compressed PostScript format. Abstracts are available from the same host in the directory `/pub/TR/UBLCS/ABSTRACTS` in plain text format. All local authors can be reached via e-mail at the address *last-name*`@cs.unibo.it`.

UBLCS Technical Report Series

93-15 *Data Algorithm: A Numerical Method to Extract Shape Information from Gray Scale Images*, R. Davoli, F. Tamburini, June 1993.

93-16 *Towards Performance Evaluation in Process Algebras*, R. Gorrieri, M. Roccetti, July 1993.

93-17 *Split and ST Bisimulation Semantics*, R. Gorrieri, C. Laneve, July 1993.

93-18 *Multilanguage Interoperability*, G. Attardi, M. Gaspari, July 1993.

93-19 *HERMES: an Expert System for the Prognosis of Hepatic Diseases*, I. Bonfà, C. Maioli, F. Sarti, G.L. Milandri, P.R. Dal Monte, September 1993.

93-20 *An Information Flow Security Property for CCS*, R. Focardi, R. Gorrieri, October 1993.

93-21 *A Classification of Security Properties*, R. Focardi, R. Gorrieri, October 1993.

93-22 *Real Time Systems: A Tutorial*, F. Panzieri, R. Davoli, October 1993.

93-23 *A Scalable Architecture for Reliable Distributed Multimedia Applications*, F. Panzieri, M. Roccetti, October 1993.

93-24 *Wide-Area Distribution Issues in Hypertext Systems*, C. Maioli, S. Sola, F. Vitali, October 1993.

93-25 *On Relating Some Models for Concurrency*, P. Degano, R. Gorrieri, S. Vigna, October 1993.

93-26 *Axiomatising ST Bisimulation Equivalence*, N. Busi, R. van Glabbeek, R. Gorrieri, December 1993.

93-27 *A Theory of Processes with Durational Actions*, R. Gorrieri, M. Roccetti, E. Stancampiano, December 1993.

94-1 *Further Modifications to the Dexter Hypertext Reference Model: a Proposal*, W. Penzo, S. Sola, F. Vitali, January 1994.

94-2 *Symbol-Level Requirements for Agent-Level Programming*, M. Gaspari, E. Motta, February 1994.

94-3 *Extending Prolog with Data Driven Rules*, M. Gaspari, February 1994.

94-4 *Schedulability Checking of Data Flow Tasks in Hard-Real-Time Distributed Systems*, R. Davoli, L. A. Giachini, March 1994.

94-5 *A Shared Dataspace Language and its Compilation*, M. Gaspari, March 1994.

94-6 *Exploring the Coordination Space with LO*, S. Castellani, P. Ciancarini, April 1994.

94-7 *Comparative Semantics of LO*, S. Castellani, P. Ciancarini, April 1994.

94-8 *Distributed Conflicts in Communicating Systems*, N. Busi, R. Gorrieri, G. Siliprandi, April 1994.

94-9 *Pseudopolar Array Mask Algorithm for Spherical Coordinate Grid Surfaces*, A. Amoroso, G. Casciola, May 1994.

94-10 *MPA: a Stochastic Process Algebra*, M. Bernardo, L. Donatiello, R. Gorrieri, May 1994.

94-11 *Describing Queueing Systems with MPA*, M. Bernardo, L. Donatiello, R. Gorrieri, May 1994.

94-12 *Operational GSPN Semantics of MPA*, M. Bernardo, L. Donatiello, R. Gorrieri, May 1994.

94-13 *Experiments in Distributing and Coordinating Knowledge*, P. Ciancarini, May 1994.

94-14 *A Comparison of Parallel Search Algorithms Based on Tree Splitting*, P. Ciancarini, May 1994.

94-15 RELACS: *A Communications Infrastructure for Constructing Reliable Applications in Large-Scale Distributed Systems*, Ö. Babaoğlu, M.G. Baker, R. Davoli, L.A. Giachini, June 1994.

94-16 *Replicated File Management in Large-Scale Distributed Systems*, Ö. Babaoğlu, A. Bartoli, G. Dini, June 1994.

Axiomatising ST-Bisimulation Equivalence [1]

Nadia Busi [2] **Rob van Glabbeek** [3] **Roberto Gorrieri** [4]

Technical Report UBLCS-93-26

December 1993
Revised June 1994

Abstract

A simple ST operational semantics for a process algebra is provided, by defining a set of operational rules in Plotkin's style. This algebra comprises TCSP parallel composition, ACP sequential composition and a refinement operator, which is used for replacing an action with an entire process, thus permitting hierarchical specification of systems. We prove that ST-bisimulation equivalence is a congruence, resorting to standard techniques on rule formats. Moreover, we provide a set of axioms that is sound and complete with respect to ST-bisimulation. The intriguing case of the forgetful refinement (i.e. when an action is refined into the properly terminated process) is dealt with in a new, improved manner.

1. To appear in *Proceedings of the IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET '94)*, S. Miniato, Italy, June 1994. The first and third authors have been supported by CNR, MURST and Esprit project 8130 LOMAPS. The second author has been funded by ONR under grant number N00014-92-J-1974.
2. Dipartimento di Matematica, Università di Siena Via del Capitano, 15, I-53100 Siena, Italy.
3. Computer Science Department, Stanford University Stanford, CA 94305, USA.
4. Dipartimento di Matematica, Università di Bologna Piazza di Porta S. Donato 5, I-40127 Bologna, Italy.

1 INTRODUCTION

Among the non-interleaving equivalences, one of the most relevant is ST-bisimulation equivalence, originally proposed in [13] over Petri Nets, from which it takes the name (S and T are the initials of the German words for Place and Transition). This semantics drops the assumption that actions are *atomic* activities, hence, unobservable in the middle of their evolution. This is implemented by splitting every action in two distinct phases: the *beginning* and the *ending* which, instead, are considered atomic. Additionally, in order to prevent possible confusion in the presence of autoconcurrent actions, every ending phase is unambiguously related to its own beginning phase through some supplementary information. In the version of ST semantics we exploit, a causal link connects every ending phase to the corresponding beginning phase [14]. This is achieved by equipping each ending phase of an action with a "relative pointer" to indicate the number of phases which have occurred from the beginning of that action.

ST semantics expresses an elementary form of duration, as many action phases can be executed in between a beginning and its end. Moreover, ST semantics is *the* semantics for the operation of *action refinement*. This operation substitutes the execution of a process for an action, thereby introducing the possibility of relating descriptions of the same system at different levels of detail. It has been extensively studied in the semantic domain of event structures [12, 6], where an event is replaced by a whole structure. ST-bisimulation equivalence has been proved to be a congruence for this operation in [9] and the coarsest congruence contained in interleaving bisimulation in [24].

In this paper, we define an ST operational semantics in SOS style [22] for a process algebra equipped with the TCSP parallel composition operator, the ACP sequential composition operator and the refinement one. The transition system we define is labelled on action phases; hence, two agents are ST-bisimulation equivalent if their corresponding states are strongly bisimilar. The transition system specification (TSS, for short) we propose fits the *tyft* format of [16]. Therefore, we can exploit a nice result of that paper, namely that strong bisimulation is a congruence. Then, we provide a sound and complete axiomatization for ST-bisimulation: to obtain this, we conservatively extend the TSS by adding some auxiliary operators, following the lines of the algorithm defined in [1].

We claim that the operational description we provide for the operator of action refinement is in perfect agreement (apart from the way the empty refinement is dealt with) with the denotational one in terms of flow event structures described in [12]. Our claim is based on a similar proof, provided in [5], where a slightly different language and a slightly different version of ST semantics are considered. We argue that our way of dealing with the empty refinement, which can be useful in applications [12], is more satisfactory than the one of [11], and that event structures are not expressive enough to model this type of refinement.

2 LABELLED TRANSITION SYSTEMS

We recall the basic, relevant notions about transition system specification from [16], incorporating a recent improvement reported in [8, 10].

Let V be a denumerable set of variables; x, y, z are typical elements of V. A single sorted signature Σ is a pair $(F, \sharp)$, where F is a set of function names disjoint with V, and $\sharp : F \rightarrow \omega$ is a rank function which gives the arity of a function name. With $\mathbb{T}(\Sigma)$ and $T(\Sigma)$ we denote the set of open and closed terms respectively over signature Σ. $V(t) \subseteq V$ denotes the set of variables in a term t. A substitution σ is a mapping $V \to \mathbb{T}(\Sigma)$, extended homomorphically to terms.

Definition 2.1 A *transition system specification* (TSS) is a triple (Σ, A, R) with Σ a signature, A a set of labels and R a set of rules of the form $\dfrac{\{t_i \xrightarrow{a_i} t_i' \mid i \in I\}}{t \xrightarrow{a} t'}$ with I an index set, $t_i, t_i', t, t' \in \mathbb{T}(\Sigma)$, $a_i, a \in A$ for $i \in I$. ■

If r is a rule satisfying the above format, then the elements of $\{t_i \xrightarrow{a_i} t_i' \mid i \in I\}$ are called the *premises* of r and $t \xrightarrow{a} t'$ is called the *conclusion* of r. A rule of the form $\dfrac{\emptyset}{t \xrightarrow{a} t'}$ is called an axiom, which, if no confusion can arise, is also written as $t \xrightarrow{a} t'$. An expression of the form $t \xrightarrow{a} t'$ with $a \in A$ and $t, t' \in \mathbb{T}(\Sigma)$ is called a transition (labelled with a). The letters ϕ, ψ, χ are used to range over transitions. The notions 'substitution', '$V(\cdot)$' and 'closed' extend to transitions and rules as expected.

Definition 2.2 Let $P = (\Sigma, A, R)$ be a TSS. A *proof* of a transition ψ from P is a well-founded, upwardly branching tree of which the nodes are labelled by transitions $t \xrightarrow{a} t'$ with $t, t' \in \mathbb{T}(\Sigma)$ and $a \in A$, such that: the root is labelled with ψ, and if χ is the label of a node q and $\{\chi_i \mid i \in I\}$ is the set of labels of the nodes directly above q, then there is a rule $\dfrac{\{\phi_i \mid i \in I\}}{\phi}$ in R and a substitution σ such that $\chi = \sigma(\phi)$ and $\chi_i = \sigma(\phi_i)$ for $i \in I$. If there exists a proof of ψ from P, then ψ is *provable* from P (notation $P \vdash \psi$). ■

Definition 2.3 A *labelled transition system (LTS)* is a structure $(S, A, \to)$ where S is a set of *states*, A an *alphabet*, and $\to \subseteq S \times A \times S$ a *transition relation*. We write $t \xrightarrow{a} t'$ to indicate that $(t, a, t') \in \to$. ■

Definition 2.4 Let $\mathcal{A} = (S, A, \to)$ be an LTS. A relation $R \subseteq S \times S$ is a *(strong) bisimulation* if it satisfies:

- if $(s, t) \in R$ and $s \xrightarrow{a} s'$, then there exists $t' \in S$ with $t \xrightarrow{a} t'$ and $(s', t') \in R$;
- if $(s, t) \in R$ and $t \xrightarrow{a} t'$, then there exists $s' \in S$ with $s \xrightarrow{a} s'$ and $(s', t') \in R$.

Two states $s, t \in S$ are bisimilar in $\mathcal{A}$, notation $\mathcal{A} : s \leftrightarrows t$, if there exists a bisimulation containing the pair (s, t). Note that bisimilarity is an equivalence relation. ■

Definition 2.5 Let $P = (\Sigma, A, R)$ be a TSS. The transition system $TS(P)$ specified by P is the triple $TS(P) = (T(\Sigma), A, \to_P)$ where the relation $\to_P$ is defined by: $t \xrightarrow{a}_P t'$ iff $P \vdash t \xrightarrow{a} t'$.

We say that two terms $t, t' \in T(\Sigma)$ are (P-)bisimilar, notation $t \leftrightarrows_P t'$, if $TS(P) : t \leftrightarrows t'$. We write $t \leftrightarrows t'$ if it is clear from the context what P is. ■

TSSs do not always generate LTSs for which bisimulation is a congruence, but they do if their rules satisfy the format below.

Definition 2.6 Let $\Sigma = (F, \sharp)$ be a signature and let $P = (\Sigma, A, R)$ be a TSS. A rule in R

is in *tyft format* if it has the form $\dfrac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1,\ldots,x_{\sharp(f)}) \xrightarrow{a} t}$ with f a function name from F, x_i $(1 \le i \le \sharp(f))$ and y_i $(i \in I)$ are all different variables, $a_i, a \in A$ and $t_i, t \in \mathbb{T}(\Sigma)$ for $i \in I$. P is in *tyft* format if all the rules in R are in *tyft* format. ■

Theorem 2.7 Let $\Sigma = (F, \sharp)$ be a signature and $P = (\Sigma, A, R)$ a TSS. If P is in *tyft* format then strong bisimulation is a congruence for all function names in F. ■

Definition 2.8 Let $P = (\Sigma, A, R)$ be a TSS and let r be a rule in R. The *bound* variables of r are recursively defined as the ones that occur in the left hand side of the conclusion or in the right hand side of a premise $t \xrightarrow{a} t'$, where t contains bound variables only. A rule r is *pure* if all variables that occur in it are bound. The TSS P is called pure if all rules in R are pure. ■

Given two TSSs P_0 and P_1 we use $P_0 \oplus P_1$ to denote their componentwise union, which is only defined if function names that occur in both the signatures of P_0 and P_1 are given the same arity. If P_0 and P_1 are in *tyft* format and some other conditions are satisfied, we have that the outgoing transitions in the transition system defined by P_0 of terms in the signature of P_0 are the same as the outgoing transitions of these terms in the transition system defined by $P_0 \oplus P_1$.

Definition 2.9 Let $P_i = (\Sigma_i, A_i, R_i)$ $(i = 0, 1)$ be two TSSs with $P = P_0 \oplus P_1$ defined. Let $P = (\Sigma, A, R)$. We say that P is a *conservative extension* of P_0 and that P_1 can be added conservatively to P_0 if for all $t \in T(\Sigma_0)$, $a \in A$ and $t' \in T(\Sigma)$:

$$P \vdash t \xrightarrow{a} t' \iff P_0 \vdash t \xrightarrow{a} t' \qquad ■$$

Note that if P is a conservative extension of P_0, P is also a conservative extension up to bisimulation, i.e. for $t, t' \in T(\Sigma_0)$: $t \leftrightarrows_P t' \iff t \leftrightarrows_{P_0} t'$.

Theorem 2.10 Let $P_0 = (\Sigma_0, A_0, R_0)$ be a TSS in pure *tyft* format and let $P_1 = (\Sigma_1, A_1, R_1)$ be a TSS in *tyft* format such that no rule of R_1 contains a function name from Σ_0 in the source of its conclusion. Let $P = P_0 \oplus P_1$ be defined. Then P_1 can be added conservatively to P_0. ■

3 THE LANGUAGE

Let *Act* be a countable set of actions; a, b, c, d range over *Act* and S over the subsets of *Act*. The process terms are generated by the following syntax:

$$t ::= \varepsilon \mid \delta \mid a \mid t \cdot t' \mid t + t' \mid t \,\|_S\, t' \mid t[a \rightsquigarrow t']$$

Intuitively, ε is the successfully terminated process, whereas δ is the deadlocked process. The operators $\cdot$ and $+$ are the ACP sequential and alternative composition, respectively. With $\|_S$ we mean the TCSP parallel composition, where synchronisation over actions in S is required[5]. When set S is empty (i.e. when no synchronization is allowed), we usually omit the subscript S in $\|_S$. Finally $[a \rightsquigarrow]$ is the refinement operator as defined e.g. in [7].

Recursion can be added and modelled in our operational approach in the usual way (see [15] for a possible operational ST semantics for recursion). However, as our main concern is the axiomatisation, we prefer to restrict our attention to the finite calculus above.

4 OPERATIONAL SEMANTICS

To define the operational semantics, we need a more generous signature to represent the states. Let k range over integers. Let $\Sigma = (F, \natural)$, where

$$\begin{array}{lcl} F & = & \{\varepsilon, \delta, +, \cdot\} \cup \{a \mid a \in Act\} \cup \{a^1 \mid a \in Act\} \cup \{[k] \mid k \neq 0\} \\ & & \cup \{\ \|_S \mid S \subseteq Act\} \cup \{[a \rightsquigarrow] \mid a \in Act\} \cup \{\overset{k}{\bowtie} \mid k > 0\} \\ \multicolumn{3}{l}{\natural(\varepsilon) = \natural(\delta) = \natural(a) = \natural(a^1) = 0} \\ \multicolumn{3}{l}{\natural([k]) = 1} \\ \multicolumn{3}{l}{\natural(+) = \natural(\cdot) = \natural(\ \|_S\ ) = \natural([a \rightsquigarrow]) = \natural(\overset{k}{\bowtie}) = 2} \end{array}$$

As any action is split in two phases, we need a unary operator to represent the state of its intermediate execution. An action a performs its beginning (denoted, with abuse of notation, a as well), reaching a^1. This performs the ending of the action a, where the superscript 1 states that this ending refers to the just performed transition. In general, the causal pointer from an ending to its beginning is implemented as a superscript number k stating that its beginning has been executed k transitions before. The *delay* operator $[k]$ suitably updates the pointers. It will be introduced in a context of parallel composition and refinement. The *merge* operator $\overset{k}{\bowtie}$ is used when merging the executions of a refined agent and the activated refining agent. Intuitively, it is a restricted form of parallel composition, as illustrated in Section 6.

Let $Act^\omega = \{\mu^k \mid \mu \in Act \wedge k > 0\}$ be the set of *endings*; let $Act_{ST} = Act \cup Act^\omega$ be the set of *phases*, where a label in Act is called a *beginning*; η ranges over Act_{ST} and θ ranges over $Act_{ST} \cup \{\surd\}$, where action $\surd$ (sometimes called "tick") denotes clean termination.

Let $P = (\Sigma, A, R)$ be the TSS where $A = Act_{ST} \cup \{\surd\}$ and R is the set of rules listed in Table 2 commented upon below. The auxiliary functions used in the rules are defined in Table 1. The functions name and index are intuitively clear. They extract from a label the name of the action and the associated index, respectively. Function sg(k) returns the sign of the integer k, while inc(k) increases the absolute value of k. Finally, function f updates the index of the label to which it is applied.[6]

Axiom $(ACT1)$ represents the starting of an action, while $(ACT2)$ accounts for the execution of the ending. $(TICK)$ is the rule for the term ε, which performs $\surd$ and terminates. The rules for sequential and alternative composition are standard (see e.g. [16]). The rules for the TCSP-like parallel composition are almost standard. According to $(PAR1)$, the asynchronous execution of a phase from the left subagent can be performed if this phase is part of an action not in the synchronisation set S; note, however, that the right subagent is

5. One could consider also more general parallel operators, like the ACP one. Our work can be easily adapted to ACP and similar languages.
6. From the definition of f we cannot exclude to obtain action endings with a zero or negative index. However, we have that terms reachable from elements of the language contain positive indexes only.

$\text{name}(a) = a$	$\text{index}(a) = 0$	$f(a, k) = a$
$\text{name}(a^k) = a$	$\text{index}(a^k) = k$	$f(a^h, k) = \begin{cases} a^{h+sg(k)} & \text{if } h \geq \lvert k \rvert \\ a^h & \text{otherwise} \end{cases}$
$\text{name}(\surd) = \surd$		$f(\surd, k) = \surd$
$\text{sg}(k) = \begin{cases} 1 & \text{if } k > 0 \\ -1 & \text{if } k < 0 \end{cases}$	$\text{inc}(k) = k + sg(k)$	

Table 1. Auxiliary functions on action phases

delayed (i.e. the delay [1] operator is applied to it), because the pointers of all ending phases it will execute that point back to before the current state must be incremented by one, as they refer to transitions which are one step further back. Rule $(PAR3)$ needs no delay as both subagents take part in the synchronisation; note that the termination signal $\surd$ can only be performed synchronously by the two subagents. The rule for the delay operator might appear a bit clumsy, as it works for integers k which can also be negative. A negative delay is needed when skipping some transitions, as now the pointers are to be decreased. The operator $[k]$ increments pointers by 1 if k is positive; it decrements by 1 if k is negative. Thus decrementing by 2 needs to be expressed as $[-1][-1]$; not as $[-2]$. The value of $|k|$ specifies *which* pointers need to be adapted; namely all pointers that point back at least k phases before the current state.

More interesting are the rules for refinement. Rule $(REF1)$ accounts for the case when the performed phase belongs to an action which is not to be refined. On the contrary, $(REF2)$ states that whenever the agent under refinement, x, starts the execution of the action to be refined, a, then $x[a \rightsquigarrow y]$ performs the first, non tick, phase the refining agent y is able to do; the reached state is the merge of the two subagents, where the parameter 1 of the merge operator remembers the index of the ending phase of the refined action. Particular care should be devoted to rule $(REF3)$ which deals with the case when y can properly terminate, as in the so-called *empty* refinement $[a \rightsquigarrow \varepsilon]$ (see Section 6 for a longer discussion). If x is ready to start the execution of a, then it will complete it, becoming x'', and then we will take the first transition the refinement of x'' is able to do, remembering to update the pointers, as we have skipped two transitions (the one for a and the other for a^1). The three rules for the merge operator state that the refined agent x can always perform a phase asynchronously, provided that this phase is not the ending of the action which has been refined (see rule $(MERGE1)$ with its side condition); that the refining agent can proceed without any constraint (see rule $(MERGE2)$); and that the merged execution of the two ends when x is ready to perform the ending of the refined action and y can terminate properly.

It is worth-while observing that the transitions are labelled with parameters that range over actions and natural numbers, and not directly with actions, as required by the *tyft* format; so, every rule in our TSS is a rule schema corresponding to an infinite set of rules.

Proposition 4.1 P is pure and in *tyft* format. Strong bisimulation is a congruence for all

$$(ACT1) \quad a \xrightarrow{a} a^1 \qquad (ACT2) \quad a^1 \xrightarrow{a^1} \varepsilon$$

$$(TICK) \quad \varepsilon \xrightarrow{\surd} \delta \qquad (DELAY) \quad \frac{x \xrightarrow{\theta} x'}{[k]x \xrightarrow{f(\theta,k)} [\mathrm{inc}(k)]x'}$$

$$(SEQ1) \quad \frac{x \xrightarrow{\eta} x'}{x \cdot y \xrightarrow{\eta} x' \cdot y} \qquad (SEQ2) \quad \frac{x \xrightarrow{\surd} x' \; y \xrightarrow{\theta} y'}{x \cdot y \xrightarrow{\theta} y'}$$

$$(ALT1) \quad \frac{x \xrightarrow{\theta} x'}{x + y \xrightarrow{\theta} x'} \qquad (ALT2) \quad \frac{y \xrightarrow{\theta} y'}{x + y \xrightarrow{\theta} y'}$$

$$(PAR1) \quad \frac{x \xrightarrow{\eta} x'}{x \parallel_S y \xrightarrow{\eta} x' \parallel_S [1]y} \quad \mathrm{name}(\eta) \notin S$$

$$(PAR2) \quad \frac{y \xrightarrow{\eta} y'}{x \parallel_S y \xrightarrow{\eta} [1]x \parallel_S y'} \quad \mathrm{name}(\eta) \notin S$$

$$(PAR3) \quad \frac{x \xrightarrow{\theta} x' \; y \xrightarrow{\theta} y'}{x \parallel_S y \xrightarrow{\theta} x' \parallel_S y'} \quad \mathrm{name}(\theta) \in S \cup \{\surd\}$$

$$(REF1) \quad \frac{x \xrightarrow{\theta} x'}{x[a \leadsto y] \xrightarrow{\theta} x'[a \leadsto y]} \quad \theta \neq a$$

$$(REF2) \quad \frac{x \xrightarrow{a} x' \; y \xrightarrow{\eta} y'}{x[a \leadsto y] \xrightarrow{\eta} x'[a \leadsto y] \overset{1}{\bowtie} y'}$$

$$(REF3) \quad \frac{x \xrightarrow{a} x' \; x' \xrightarrow{a^1} x'' \; [-1][-1](x''[a \leadsto y]) \xrightarrow{\theta} x''' \; y \xrightarrow{\surd} y'}{x[a \leadsto y] \xrightarrow{\theta} x'''}$$

$$(MERGE1) \quad \frac{x \xrightarrow{\eta} x'}{x \overset{k}{\bowtie} y \xrightarrow{\eta} x' \overset{k+1}{\bowtie} [1]y} \quad \mathrm{index}(\eta) \neq k$$

$$(MERGE2) \quad \frac{y \xrightarrow{\eta} y'}{x \overset{k}{\bowtie} y \xrightarrow{\eta} [1]x \overset{k+1}{\bowtie} y'}$$

$$(MERGE3) \quad \frac{x \xrightarrow{a^k} x' \; [-1]x' \xrightarrow{\theta} x'' \; y \xrightarrow{\surd} y'}{x \overset{k}{\bowtie} y \xrightarrow{\theta} x''}$$

Table 2. Rules for basic operators

function names in F. ■

5 AXIOMATISATION

In this section we introduce a set of auxiliary operators; then we define an axiomatisation and prove its soundness and completeness with respect to bisimulation.

5.1 Auxiliary Operators

First we introduce a set of auxiliary operators, that are necessary to obtain the axiomatisation. Let $\Sigma' = (F', \sharp')$, where:

$$\begin{array}{lcl} F' & = & F \cup \{a^+ \mid a \in Act\} \cup \{a^k \mid a \in Act \wedge k \geq 2\} \\ & & \cup \{ \lfloor\!\lfloor_S \,, \,|_S \mid S \subseteq Act\} \cup \{\overset{k}{\lhd}, \overset{k}{\rhd} \mid k > 0\} \cup \{\gg\} \\ \multicolumn{3}{l}{\text{if } f \in F \text{ then } \sharp'(f) = \sharp(f)} \\ \multicolumn{3}{l}{\sharp'(a^+) = \sharp'(a^k) = 0} \\ \multicolumn{3}{l}{\sharp'(\lfloor\!\lfloor_S) = \sharp'(|_S) = \sharp'(\overset{k}{\lhd}) = \sharp'(\overset{k}{\rhd}) = \sharp'(\gg) = 2} \end{array}$$

In this setting, it is essential to introduce a syntactical distinction between actions and beginnings. To this aim, we introduce the auxiliary beginnings of the form a^+, which can perform the beginning a and then terminate successfully. The operators of the form a^k, for any $k \geq 2$, are needed as we have labels of this kind in the operational rules of the previous section. The left-parallel $\lfloor\!\lfloor_S$ and the synchronisation-parallel $|_S$ are useful for the axiomatisation of the parallel operator $\|_S$. They play the same rôle as the left-merge and communication merge of ACP [3]. However, whereas the left-merge of [3] insists that the first *action* in a parallel composition comes from the argument on the left, our $\lfloor\!\lfloor_S$ only insists that the first *phase* comes from the argument on the left. As far as the mnemonics concerns, $\lfloor\!\lfloor_S$ points to the *left*, whereas the left merge is shaped like an L; there is no need for a right-merge here. The auxiliary operators $\overset{k}{\lhd}$ and $\overset{k}{\rhd}$ play a similar rôle in the axiomatisation of the $\overset{k}{\bowtie}$. Due to the asymmetry between the refined and the refining process both are needed here. $\gg$ is used to axiomatise the refinement operator.

Let $P' = (\Sigma', A, R')$ a TSS, where R' is the set of rules listed in Tables 2 and 3. The only rules we comment on are those for $\gg$. The first rule states that the next transition, if not tick, must be taken from the right subagent; the reached state is the merge of the two, with pointer one. The second rule, instead, covers the case when y can terminate. This rule is similar to $(REF3)$. In this way $x \gg y$ behaves exactly like $x[a \rightsquigarrow y]$ when the latter is about to start refining an action a.

Proposition 5.1 P' is in *tyft* format. Strong bisimulation is a congruence for all function names in F'. P' is a conservative extension of P. ■

The binary operators are left associative and binding priority is defined in decreasing order as follows: $\cdot > [k] > [a \rightsquigarrow] > \|_S \,, \lfloor\!\lfloor_S \,, |_S \,, \overset{k}{\bowtie}, \overset{k}{\lhd}, \overset{k}{\rhd}, \gg\, > +$.

$$(ACT3)\quad a^+ \xrightarrow{a} \varepsilon \qquad (ACT4)\quad a^k \xrightarrow{a^k} \varepsilon \quad k > 1$$

$$(RTOM1)\quad \frac{y \xrightarrow{\eta} y'}{x \gg y \xrightarrow{\eta} x \overset{1}{\bowtie} y'} \qquad (RTOM2)\quad \frac{x \xrightarrow{a^1} x' \; [-1][-1]x' \xrightarrow{\theta} x'' \; y \xrightarrow{\surd} y'}{x \gg y \xrightarrow{\theta} x''}$$

$$(RME1)\quad \frac{y \xrightarrow{\eta} y'}{x \overset{k}{\rhd} y \xrightarrow{\eta} [1]x \overset{k+1}{\bowtie} y'} \qquad (RME2)\quad \frac{x \xrightarrow{a^k} x' \; [-1]x' \xrightarrow{\theta} x'' \; y \xrightarrow{\surd} y'}{x \overset{k}{\rhd} y \xrightarrow{\theta} x''}$$

$$(LME)\quad \frac{x \xrightarrow{\eta} x'}{x \overset{k}{\lhd} y \xrightarrow{\eta} x' \overset{k+1}{\bowtie} [1]y} \quad \text{index}(\eta) \neq k$$

$$(LPAR)\quad \frac{x \xrightarrow{\eta} x'}{x \;⫫_S\; y \xrightarrow{\eta} x' \parallel_S [1]y} \quad \text{name}(\eta) \notin S$$

$$(SYNC)\quad \frac{x \xrightarrow{\theta} x' \; y \xrightarrow{\theta} y'}{x \mid_S y \xrightarrow{\theta} x' \parallel_S y'} \quad \text{name}(\theta) \in S \cup \{\surd\}$$

Table 3. Rules for auxiliary operators

5.2 Axiomatisation

Let $Act^+ = \{\mu^+ \mid \mu \in Act\}$. Let λ range over $Act^+ \cup Act^\omega \cup \{\delta\}$. We extend the auxiliary functions introduced in the previous section in the following way:

name$(a^+) = a$	index$(a^+) = 0$	$f(a^+, k) = a^+$
name$(\delta) = \delta$	index$(\delta) = 0$	$f(\delta, k) = \delta$

Let $\mathcal{T}$ be the axiom set in Tables 4 and 5. Our axioms for the interaction between $\|$ and the empty process is a mild variation on the treatment of [25]. While the axioms for the alternative, sequential and parallel composition are almost standard, the other axioms are not so usual. First, observe axiom (act). It states that any action a should be intended as composed of the sequence of its beginning and its ending (with pointer 1). The three axioms for the delay operator reflect clearly its operational definition. More intriguing is the case of refinement; in particular, $(ref2)$ shows that the refinement can start by dropping the beginning of the refined action and giving priority to the refining subagent y. Note that no action has been extracted from the term, so, no phase has been "executed". This is done by the first application of axiom $(rtom1)$, if y is not terminated, or of $(rtom2)$, if y terminates properly. The axioms for $\overset{k}{\lhd}$ and $\overset{k}{\rhd}$ are intuitively clear, as $(lmerge1)$ corresponds to the operational rule $(MERGE1)$, $(rmerge1)$ to $(MERGE2)$ and $(rmerge2)$ to $(MERGE3)$. Finally, note that the axioms for $\gg$ and $\overset{k}{\rhd}$ are very similar: they differ only in the application of the delay operator in the first two axioms.

5.3 Soundness and Completeness

We cannot apply the algorithm presented in [1] because the rules of our TSS do not fit the GSOS format used in that paper, due to the presence in our TSS of lookahead and of an

$(alt1)$	$x + y$	$=$	$y + x$	$(seq1)$	$(xy)z$	$=$	$x(yz)$
$(alt2)$	$(x + y) + z$	$=$	$x + (y + z)$	$(seq2)$	$(x + y)z$	$=$	$xz + yz$
$(alt3)$	$x + x$	$=$	x	$(seq3)$	δx	$=$	δ
$(alt4)$	$x + \delta$	$=$	x	$(seq4)$	εx	$=$	x
				$(seq5)$	$x\varepsilon$	$=$	x
(act)	a	$=$	$a^+ a^1$				
(par)	$x \parallel_S y$	$=$	$x \mathbin{⫡}_S y + y \mathbin{⫡}_S x + x \mid_S y$				
$(lpar1)$	$\lambda x \mathbin{⫡}_S y$	$=$	$\lambda(x \parallel_S [1]y)$	if name$(\lambda) \notin S$			
$(lpar2)$	$\lambda x \mathbin{⫡}_S y$	$=$	δ	if name$(\lambda) \in S$			
$(lpar3)$	$\varepsilon \mathbin{⫡}_S x$	$=$	δ				
$(lpar4)$	$(x + y) \mathbin{⫡}_S z$	$=$	$x \mathbin{⫡}_S z + y \mathbin{⫡}_S z$				
$(sync1)$	$\lambda x \mid_S \lambda y$	$=$	$\lambda(x \parallel_S y)$	if name$(\lambda) \in S$			
$(sync2)$	$\lambda x \mid_S \lambda' y$	$=$	δ	if name$(\lambda) \notin S \vee \lambda \neq \lambda'$			
$(sync3)$	$\varepsilon \mid_S \lambda y$	$=$	δ				
$(sync4)$	$\varepsilon \mid_S \varepsilon$	$=$	ε				
$(sync5)$	$(x + y) \mid_S z$	$=$	$x \mid_S z + y \mid_S z$				
$(sync6)$	$x \mid_S y$	$=$	$y \mid_S x$				
$(delay1)$	$[k](\lambda x)$	$=$	$f(\lambda, k)([\text{inc}(k)]x)$				
$(delay2)$	$[k]\varepsilon$	$=$	ε				
$(delay3)$	$[k](x + y)$	$=$	$[k]x + [k]y$				

Table 4. Axioms for all the operators (except refinement)

$(ref1)$	$(\lambda x)[a \leadsto y]$	$=$	$\lambda(x[a \leadsto y])$	if $\lambda \neq a^+$
$(ref2)$	$(a^+ x)[a \leadsto y]$	$=$	$(x[a \leadsto y]) \gg y$	
$(ref3)$	$\varepsilon[a \leadsto y]$	$=$	ε	
$(ref4)$	$(x+y)[a \leadsto z]$	$=$	$x[a \leadsto z] + y[a \leadsto z]$	
$(rtom1)$	$x \gg \lambda y$	$=$	$\lambda(x \overset{1}{\bowtie} y)$	
$(rtom2)$	$a^1 x \gg \varepsilon$	$=$	$[-1][-1]x$	
$(rtom3)$	$\lambda x \gg \varepsilon$	$=$	δ	if index$(\lambda) \neq 1$
$(rtom4)$	$\varepsilon \gg \varepsilon$	$=$	δ	
$(rtom5)$	$(x+y) \gg \varepsilon$	$=$	$x \gg \varepsilon + y \gg \varepsilon$	
$(rtom6)$	$x \gg (y+z)$	$=$	$x \gg y + x \gg z$	
$(merge)$	$x \overset{k}{\bowtie} y$	$=$	$x \overset{k}{\lhd} y + x \overset{k}{\rhd} y$	
$(lmerge1)$	$\lambda x \overset{k}{\lhd} y$	$=$	$\lambda(x \overset{k+1}{\bowtie} [1]y)$	if index$(\lambda) \neq k$
$(lmerge2)$	$\lambda x \overset{k}{\lhd} y$	$=$	δ	if index$(\lambda) = k$
$(lmerge3)$	$\varepsilon \overset{k}{\lhd} x$	$=$	δ	
$(lmerge4)$	$(x+y) \overset{k}{\lhd} z$	$=$	$x \overset{k}{\lhd} z + y \overset{k}{\lhd} z$	
$(rmerge1)$	$x \overset{k}{\rhd} \lambda y$	$=$	$\lambda([1]x \overset{k+1}{\bowtie} y)$	
$(rmerge2)$	$a^k x \overset{k}{\rhd} \varepsilon$	$=$	$[-1]x$	
$(rmerge3)$	$\lambda x \overset{k}{\rhd} \varepsilon$	$=$	δ	if index$(\lambda) \neq k$
$(rmerge4)$	$\varepsilon \overset{k}{\rhd} \varepsilon$	$=$	δ	
$(rmerge5)$	$(x+y) \overset{k}{\rhd} \varepsilon$	$=$	$x \overset{k}{\rhd} \varepsilon + y \overset{k}{\rhd} \varepsilon$	
$(rmerge6)$	$x \overset{k}{\rhd} (y+z)$	$=$	$x \overset{k}{\rhd} y + x \overset{k}{\rhd} z$	

Table 5. Axioms for the refinement operator

infinite set of rules and operators. However, we follow the lines presented there.

Given a set E of equations between terms, and $t = t'$ another equation, we write $E \vdash t = t'$ to indicate that the equation $t = t'$ is derivable from E by means of equational logic. The rules of equational logic assert that equality is a congruence that it is preserved by instantiation of variables. For a formal definition see, e.g. [17].

In the following we state that the theory $\mathcal{T}$ is sound and complete with respect to $\leftrightarrows$.

Theorem 5.2 If $(t = t') \in \mathcal{T}$ then for any closed substitution ρ we have $\rho(t) \leftrightarrows \rho(t')$.

Proof. We report the proofs for a selected set of significant axioms.
Let ρ be a closed substitution.
(ref2) We want to prove that $\rho((a^+x)[a \rightsquigarrow y]) \leftrightarrows \rho((x[a \rightsquigarrow y]) \gg y)$, that is $(a^+\rho(x))[a \rightsquigarrow \rho(y)] \leftrightarrows (\rho(x)[a \rightsquigarrow \rho(y)]) \gg \rho(y)$.

If $(a^+\rho(x))[a \rightsquigarrow \rho(y)] \xrightarrow{\theta} t$ three cases may occur.

- The transition is obtained by applying $(REF1)$. Then the transition $a^+\rho(x) \xrightarrow{\theta}$ is derivable, with $\theta \neq a$. This transition can be obtained in two ways:
  by $(SEQ1)$, so we need that $a^+ \xrightarrow{\theta}$, but the only transition for a^+ is $a^+ \xrightarrow{a} \epsilon$, hence $\theta = a$ (contradiction); or
  by $(SEQ2)$, but we need that $a^+ \xrightarrow{\surd}$ (contradiction).
- The transition is obtained by applying $(REF2)$. Then we have that $\theta \neq \surd$, $a^+\rho(x) \xrightarrow{a} u$, $\rho(y) \xrightarrow{\theta} v$ and $t = u[a \rightsquigarrow \rho(y)] \overset{1}{\bowtie} v$. The transition $a^+\rho(x) \xrightarrow{a} u$ can be obtained only by an application of $(ACT3)$ and $(SEQ1)$, so we have that $u = \varepsilon\rho(x)$, from which $t = \varepsilon\rho(x)[a \rightsquigarrow \rho(y)] \overset{1}{\bowtie} v$. Applying $(RTOM1)$ to the transition $\rho(y) \xrightarrow{\theta} v$ (remember that $\theta \neq \surd$) we obtain $(\rho(x)[a \rightsquigarrow \rho(y)]) \gg \rho(y) \xrightarrow{\theta} (\rho(x)[a \rightsquigarrow \rho(y)]) \overset{1}{\bowtie} v$. From the soundness of axiom $(seq4)$ we have that $\varepsilon\rho(x) \leftrightarrows \rho(x)$, thus from the congruence property of $\leftrightarrows$ we have $t \leftrightarrows (\rho(x)[a \rightsquigarrow \rho(y)]) \overset{1}{\bowtie} v$.
- The transition is obtained by applying $(REF3)$. Then we have $a^+\rho(x) \xrightarrow{a} u$, $u \xrightarrow{a^1} u'$, $[-1][-1](u'[a \rightsquigarrow \rho(y)]) \xrightarrow{\theta} t$ and $\rho(y) \xrightarrow{\surd} v$. As in the previous case, we have $u = \varepsilon\rho(x)$. The only explanation for the transition $\varepsilon\rho(x) \xrightarrow{a^1} u'$ is through $(SEQ2)$ and $(TICK)$, thus it must be that $\rho(x) \xrightarrow{a^1} u'$. Being $a^1 \neq a$, with an application of $(REF1)$ to this transition we obtain $\rho(x)[a \rightsquigarrow \rho(y)] \xrightarrow{a^1} u'[a \rightsquigarrow \rho(y)]$. Applying $(RTOM2)$ yields $(\varepsilon\rho(x)[a \rightsquigarrow \rho(y)]) \gg \rho(y) \xrightarrow{\theta} t$.

Symmetrically, we obtain that if $(\rho(x)[a \rightsquigarrow \rho(y)]) \gg \rho(y) \xrightarrow{\theta} t$ there exists an u such that $(a^+\rho(x))[a \rightsquigarrow \rho(y)] \xrightarrow{\theta} u$ and $t \leftrightarrows u$.
(delay1) We want to prove that $[k](\lambda\rho(x)) \leftrightarrows f(\lambda, k)([\mathrm{inc}(k)]\rho(x))$.

Suppose that $[k](\lambda\rho(x)) \xrightarrow{\theta} t$. This transition can be obtained only by $(DELAY)$; so there exist θ' and u such that $\lambda\rho(x) \xrightarrow{\theta'} u$, $\theta = f(\theta', k)$ and $t = [\mathrm{inc}(k)]u$. From $\lambda\rho(x) \xrightarrow{\theta'} u$ we can derive that $\lambda \neq \delta$, $\theta' = \lambda$ and $u = \varepsilon\rho(x)$. Hence $f(\lambda, k) = \theta$ and with $(ACT3$ or $4)$ and $(SEQ1)$ we obtain that $f(\lambda, k)([\mathrm{inc}(k)]\rho(x)) \xrightarrow{\theta} \varepsilon([\mathrm{inc}(k)]\rho(x))$. From the soundness of axiom $(seq4)$ and the congruence property of $\leftrightarrows$ we have that $[\mathrm{inc}(k)]\varepsilon\rho(x) \leftrightarrows \varepsilon([\mathrm{inc}(k)]\rho(x))$.

The other direction of the proof is similar.
(rtom2) We want to prove that $(a^1\rho(x)) \gg \varepsilon \leftrightarrows [-1][-1]\rho(x)$.

Suppose that $(a^1\rho(x)) \gg \varepsilon \xrightarrow{\theta} t$. Because the only possible transition for ε is $\varepsilon \xrightarrow{\surd} \delta$, the transition above cannot be obtained by an application of the rule ($RTOM1$). Hence ($RTOM2$) has been applied. As the only transition for $a^1\rho(x)$ is $a^1\rho(x) \xrightarrow{a^1} \varepsilon\rho(x)$ it must be that $[-1][-1]\varepsilon\rho(x) \xrightarrow{\theta} t$. From the soundness of axiom (*seq*4) and the congruence property of $\leftrightarrows$ we have that $[-1][-1]\varepsilon\rho(x) \leftrightarrows [-1][-1]\rho(x)$, so we have that $[-1][-1]\rho(x) \xrightarrow{\theta} u$ and $t \leftrightarrows u$.

Again the other direction of the proof is similar. ■

Corollary 5.3 If $\mathcal{T} \vdash t = t'$ then for any closed substitution ρ we have $\rho(t) \leftrightarrows \rho(t')$. ■

Definition 5.4 A term $t \in T(\Sigma')$ is in *normal form* if it can be generated by the following grammar:

$$n ::= \varepsilon \mid \delta \mid a^+ \cdot n \mid a^k \cdot n \mid n + n'$$

A term $t \in T(\Sigma')$ can be *brought in normal form* if there is a term n in normal form with $\mathcal{T} \vdash t = n$. ■

Definition 5.5 Let n be a term in normal form. The length of n is defined in the following way:

$$\begin{array}{ll} l(\varepsilon) = 1 & l(a^+ \cdot n) = l(n) + 1 \\ l(\delta) = 1 & l(a^k \cdot n) = l(n) + 1 \\ l(n + n') = l(n) + l(n') + 1 & \end{array}$$

■

Lemma 5.6 Let n and m be terms in normal form. Then

1. the term $n \cdot m$ can be brought in normal form;
2. for all $k \neq 0$ there exists a term n' in normal form with $\mathcal{T} \vdash [k]n = n'$ and $l(n') = l(n)$;
3. the terms $n \amalg_S m$, $n \mid_S m$ and $n \parallel_S m$ can be brought in normal form;
4. for all $k > 0$ the terms $n \overset{k}{\triangleright} m$, $n \overset{k}{\triangleleft} m$ and $n \overset{k}{\bowtie} m$ can be brought in normal form;
5. the term $n \gg m$ can be brought in normal form;
6. and the term $n[a \rightsquigarrow m]$ can be brought in normal form.

Proof. We establish part 4 with induction on the sum of the length of n and m. The other parts proceed likewise.

Let n and m be two terms in normal form and $l(n) + l(m) = i$. We first prove that there exists a term p in normal form such that $\mathcal{T} \vdash n \overset{k}{\triangleright} m = p$.

- if m is ε we can have the following cases:
  - if n is ε, we have that $\varepsilon \overset{k}{\triangleright} \varepsilon = \delta$ (*rmerge*4), and δ is in normal form.
  - if n is δ, we use that $\delta = \delta x$ (seq3). Thus, using (*rmerge*3) we have $\delta \overset{k}{\triangleright} \varepsilon = \delta x \overset{k}{\triangleright} \varepsilon = \delta$.
  - if n is $a^+ n'$ we have $a^+ n' \overset{k}{\triangleright} \varepsilon = \delta$, by (*rmerge*3).
  - if n is $a^h n'$ and $h \neq k$, we also have $a^h n' \overset{k}{\triangleright} \varepsilon = \delta$.
  - if n is $a^k n'$, we have $a^k n' \overset{k}{\triangleright} \varepsilon = [-1]n'$ by (*rmerge*2), on which we apply part 2 of the lemma.

  - if n is $n'+n''$, we have $(n'+n'')\overset{k}{\rhd}\varepsilon = n'\overset{k}{\rhd}\varepsilon+n''\overset{k}{\rhd}\varepsilon$ by $(rmerge5)$. As $l(n'), l(n'') < l(n)$, we may apply the induction hypothesis, so there must be terms p' and p'' in normal form such that $n'\overset{k}{\rhd}\varepsilon = p'$ and $n''\overset{k}{\rhd}\varepsilon = p''$. It follows that $(n'+n'')\overset{k}{\rhd}\varepsilon = p'+p''$.
- if m is δ: $n\overset{k}{\rhd}\delta \overset{seq3}{=} n\overset{k}{\rhd}(\delta x) \overset{rmerge1}{=} \delta([1]n\overset{k+1}{\bowtie}x) \overset{seq3}{=} \delta$.
- if m is a^+m': by $(rmerge1)$ we obtain $n\overset{k}{\rhd}(a^+m') = a^+([1]n\overset{k+1}{\bowtie}m')$. By part 2 of the lemma there exists a term n' in normal form such that $[1]n = n'$ and $l(n) = l(n')$. As equality is a congruence we have $[1]n\overset{k+1}{\bowtie}m' = n'\overset{k+1}{\bowtie}m'$. Moreover, $l(n')+l(m') < i$, so, by induction, there exists a term p' in normal form such that $n'\overset{k+1}{\bowtie}m' = p'$; it follows that $n\overset{k}{\rhd}a^+m' = a^+p'$, where a^+p' is in normal form.
- if m is a^km', the proof is equal to the above one.
- if m is $m'+m''$, by $(rmerge6)$ we obtain $n\overset{k}{\rhd}(m'+m'') = n\overset{k}{\rhd}m' + n\overset{k}{\rhd}m''$, which by induction can be brought in normal form.

Following the same lines, we can bring $n\overset{k}{\lhd}m$ in normal form. Finally, $n\overset{k}{\bowtie}m$ is brought in normal form through an application of $(merge)$, using the normalisation of $n\overset{k}{\rhd}m$ and $n\overset{k}{\lhd}m$.
■

Theorem 5.7 Let $t \in T(\Sigma')$. There exists then a term n in normal form such that $\mathcal{T} \vdash t = n$.
■

This theorem follows by induction from lemma 5.6. Due to the presence of axioms like $(rtom2)$, lemma 5.6 cannot be replaced by a simpler lemma, only obtaining *head normal forms*, as in [1].

Lemma 5.8 Let n, m be closed terms in normal form. If $n \leftrightarrows m$ then $\mathcal{T} \vdash n = m$.

Proof. Standard (see [19]). ■

Corollary 5.9 Let $t, t' \in T(\Sigma')$. If $t \leftrightarrows t'$ then $\mathcal{T} \vdash t = t'$. ■

6 CONCLUDING REMARKS

In this section, we discuss some further issues. First we provide an alternative operational description of refinement which exploits the parallel composition operator (with synchronisation allowed on a special end action e only). Then, we compare the various approaches to the empty refinement (i.e. when the refining agent is properly terminated) that appear in the literature. Finally, we comment on related work on other operational versions of ST semantics and definitions of action refinement.

6.1 Coding Refinement with Parallel Composition

The operational description of action refinement and its axiomatisation we provide in Sections 4 and 5 have the merits of being "context-free", in the sense that these rules and axioms can be "plugged in" in any other operational and axiomatic description of a process algebra, provided that this algebra is equipped with choice and action-prefix operators.

Anyway, it is easy to see that the merge operator is indeed a kind of parallel composition, preventing synchronisations between the refined agent and the refining one. Hence, with the aim at minimising the number of rules and axioms, here we present an alternative description of action refinement which implements the merge through a slight modification of parallel composition, using also the sequential composition operator. In the following, operators $\overset{k}{\bowtie}$, $\gg$, $\overset{k}{\triangleright}$ and $\overset{k}{\triangleleft}$ are no longer used. On the other hand, we introduce a new constant e, exploited to signal termination of the refining agent (not to be confused with $\surd$), and a new unary operator skip which "skips" the transitions labelled by e.

The idea consists in replacing the ending phase of the currently refined action with the special action e, which will be added also at the end of the refining term. Then we put in parallel the remaining part of the refined term with the refining one, preventing any synchronisation on normal actions, but forcing the synchronisation on label e. Subsequently, the now "useless" e-labelled transitions are skipped via the new unary operator skip. In the following, labels η and θ range also over e. The rules for the new operators are listed below:

$$(END) \qquad e \xrightarrow{e} \varepsilon$$

$$(SKIP1) \qquad \frac{x \xrightarrow{\theta} x'}{\text{skip}(x) \xrightarrow{\theta} \text{skip}(x')} \quad \theta \neq e$$

$$(SKIP2) \qquad \frac{x \xrightarrow{e} x' \quad [-1]x' \xrightarrow{\theta} x''}{\text{skip}(x) \xrightarrow{\theta} x''}$$

The rules for parallel composition are to be modified slightly in order to cope with the new label e, which is always a synchronisation action (similarly to $\surd$). The other rules in the original TSS are unchanged, except for the fact that now the variables η and θ can take also the value e. The three rules for refinement are now replaced by the two rules below, where the second one covers also the case of empty refinement.

$$(REF1) \qquad \frac{x \xrightarrow{\theta} x'}{x[a \rightsquigarrow y] \xrightarrow{\theta} x'[a \rightsquigarrow y]} \qquad \theta \neq a$$

$$(REF2) \qquad \frac{x \xrightarrow{a} x' \quad \text{skip}([-1](x'[a \rightsquigarrow y]) \parallel ye) \xrightarrow{\theta} y'}{x[a \rightsquigarrow y] \xrightarrow{\theta} y'}$$

The delay rule remains the same, but function f is changed by substituting the end action e for the ending phase of the refined action.

$$f(a^h, k) = \begin{cases} a^h & \text{if } h \leq |k| - 1 \\ e & \text{if } h = -k \\ a^{h+sg(k)} & \text{otherwise} \end{cases}$$

Note that the value of $f(a^h, -h)$ is not significant in the preceding sections; if we would have had an action e, we could just as well have defined f as above from the beginning.

Also for the axiomatisation, all the axioms related to the dropped operators are removed. Moreover, the axioms for the synchronisation merge take into account the fact that e is a synchronisation action. Below the axioms for the new operators and for the new definition of action refinement are reported, where now λ can also take the value e.

$(ref1)$	$(\lambda x)[a \rightsquigarrow y]$	$=$	$\lambda(x[a \rightsquigarrow y])$ if $\lambda \neq a^+$
$(ref2)$	$(a^+x)[a \rightsquigarrow y]$	$=$	$\mathrm{skip}([-1](x[a \rightsquigarrow y]) \parallel ye)$
$(ref3)$	$\varepsilon[a \rightsquigarrow y]$	$=$	ε
$(ref4)$	$(x+y)[a \rightsquigarrow z]$	$=$	$x[a \rightsquigarrow z] + y[a \rightsquigarrow z]$
$(skip1)$	$\mathrm{skip}(\lambda x)$	$=$	$\lambda\mathrm{skip}(x)$ if $\lambda \neq e$
$(skip2)$	$\mathrm{skip}(ex)$	$=$	$[-1]x$
$(skip3)$	$\mathrm{skip}(\varepsilon)$	$=$	ε
$(skip4)$	$\mathrm{skip}(x+y)$	$=$	$\mathrm{skip}(x) + \mathrm{skip}(y)$

The reader can easily get convinced that this formulation is equivalent to the one of Sections 4 and 5.

6.2 Empty Refinements

There exists no general agreement on what an empty refinement, i.e. the refinement of an action into a successfully terminated process, should be. Our proposal differs from those that already appeared in the literature.

In [11] an interpretation of the empty refinement is offered in the domain of prime event structures, according to which events refined into the empty structure are simply erased; a consequence of this decision is that the congruence result for ST bisimulation semantics does not hold in this approach, as shown in [9]. Moreover they argue that these *forgetful refinements* can drastically change the behaviour of concurrent systems and can not be explained by a change in the level of abstraction at which these systems are regarded [12]. In [7], the empty refinement is treated as if the refining agent were deadlocked. The justification for this proposal is that empty refinement is an erroneous step in the top-down development procedure. In this way, the congruence property is kept. We could obtain the same result by skipping rule $(REF3)$.

Now we introduce an example to illustrate the differences among these interpretations of empty refinement. Let $t = (ab \parallel_b (b+c) \parallel_c cd) \parallel_{ad} ad$. In Figure 1 this expression is depicted as an event structure. The densely dotted line indicates syntactic conflict, as specified by the $+$-operator in t. The sparsely dotted line indicates inherited or semantic conflict: the events b and d cannot occur in the same run as c is a prerequisite of d and in (syntactic) conflict with b.

If we refine the action c into ε in t, following the paradigm of [7], event c is blocked and the resulting system is equivalent to $ab\delta$, corresponding with the ST transition system in Figure 2a. If we, in the spirit of [11], simply erase event c in a model of event structures (such as *flow event structures* [4]) where semantic conflict is merely a matter of interpretation, we would obtain a system that is equivalent to $a(b \| d)$, in which the conflict between b and d is dropped. However, the forgetful refinements of [11, 9] are considered on *prime* event structures [21], were semantic conflict has the same status as syntactic conflict. Here the result is $a(b+d)$, corresponding to the ST transition system in Figure 2b. A reason to dislike this result is that

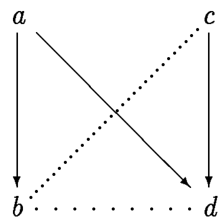

Figure 1. Event structure representation of t

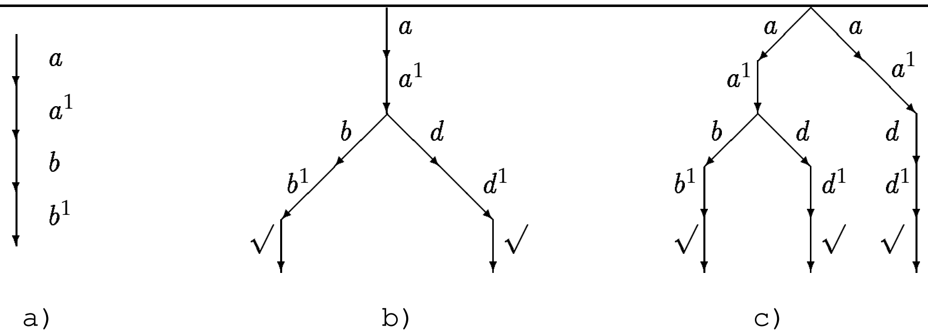

Figure 2. $t[c \rightsquigarrow \varepsilon]$ according to `a)` **[7],** `b)` **[11] and** `c)` **[25]**

the choice between b and d is made only after the a occurred, whereas in the original system a commitment to do d could be made earlier through the occurrence of c. Hence this notion of forgetful refinement can mask deadlock behaviour of concurrent systems. In [25] the empty refinement is called 'renaming into ε' and $[a \rightsquigarrow \varepsilon]$ is denoted $\varepsilon_{\{a\}}$. His solution would yield $a(b+d)+ad$, which better captures the branching structure between the executions ab and ad. But [25] works in interleaving semantics (and doesn't consider other types of action refinement) and consequently his solution cannot capture the fact that the choice between these executions can be made at any moment during the execution of a. This is captured by the operational semantics defined in Section 4. We obtain the transition system in Figure 3.

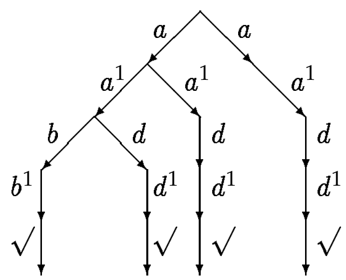

Figure 3. The ST transition system for $t[c \rightsquigarrow \varepsilon]$ according to our rules.

As this transition system cannot be obtained as the image of an event structure, it follows that event structures are not expressive enough for this type of empty refinement.

6.3 Related Work

In this section, we briefly describe some related work on alternative operational ST semantics and on alternative operational definitions of action refinement.

Different Approaches to ST Semantics We restrict our attention to those proposals for which ST-bisimulation semantics is defined as standard bisimulation on a transition system labelled on suitable phases of actions.

In [2] Aceto and Hennessy study a finite process algebra equipped with a parallel composition operator which does not permit synchronisation. For this simple language they define, as we do here, a transition system labelled on action phases, where however the endings are not provided with pointers. Standard bisimulation equivalence gives rise to an equivalence relation called by the authors *timed* equivalence, which is also known in the literature under the name of $split_2$ equivalence [9, 14]. Timed equivalence coincides with ST-bisimulation equivalence because of the limited expressive power of the language they investigate. An elegant equational characterisation of timed equivalence is provided.

An operational ST semantics for full CCS [20] is reported in [15] where the pointers from endings to beginnings are "absolute", i.e. the index k in a^k indicates that the ending refers to the k-th started action. This makes the definition of an operational semantics much more involuted because there is the need of a global counter (states are pairs $\langle t, z \rangle$, where t is a process term and z is a natural number) in order to implement the absolute referencing, and rules must be specialised to beginnings and endings (hence, duplicated) because only at beginnings pointers are increased. In this paper, no axiomatisation is proposed and action refinement is not considered.

In [18], an operational semantics with "weak" transitions (i.e. transitions corresponding to those of weak bisimulation [20]) for a CCS-like language is presented. However, the transition system generated is infinitely-branching also for finite (i.e. recursion free) agents. Moreover, the operational rules for parallel composition are context-dependent. On this basis, a non-standard proof system is proposed, composed of conditional, context-dependent equations. Also in this paper action refinement is not considered.

Other Operational Semantics for Action Refinement There are three papers dealing with this problem, the first one working with timed semantics [2], the second one with a causal semantics [7] and the last one with the even finer semantics of event isomorphism [23].

In [2] an operational semantics in SOS style for the operator of action refinement — $x[a \rightsquigarrow y]$ — is defined following tightly the inductive structure of the term x to which the operator is applied. For instance, if $x = x_1 + x_2$, then $x[a \rightsquigarrow y]$ can do whatever $x_1[a \rightsquigarrow y]$ and $x_2[a \rightsquigarrow y]$ are able to do. This definition is equivalent to making a complete syntactic subsitution of the refining agent y for any occurrence of a in x before executing the agent. (Compare the classic notion of the "dynamic" copy rule opposed to the static one for the implementation of procedure calls.) This operational semantics, however, does not scale easily to langauges which include communication and restriction; moreover, the basic intuition of refinement in such a case is no longer in full agreement with the corresponding operation on Event Structures.

In [7] a causal operational semantics is introduced for action refinement in a finite language which is very close to ours (the only difference is that we take ACP sequential composition instead of CCS action prefixing). This work relies upon previous work [6], where Darondeau and Degano prove that the inductive definition of action refinement they propose on Causal Trees coincides with the one they propose on a special subclass of Event Structures, called Δ-*free*. Our work owes some basic intuition to [7], which is similar in spirit, except for the way the empty refinement is dealt with.

In [23] Rensink shows that action refinement can receive a simple operational treatment when working with the very concrete semantics induced by event isomorphism. This definition is proved to agree with a denotational operation of refinement defined over his model of *families of posets*.

In our opinion, the main merit of our proposal w.r.t. the last two is that we (arguably) use the "right" semantics, as ST-bisimulation equivalence has been proved to be the coarsest congruence contained in interleaving bisimulation [24]. However, the other two proposals define transition systems which are smaller because actions do not have to be split in phases.

References

[1] L. Aceto, B. Bloom, and F.W. Vaandrager. "Turning SOS rules into equations". *Information and Computation*, 111(1):1–52, 1994.

[2] L. Aceto and M. Hennessy. "Towards action-refinement in process algebras". *Information and Computation*, 103(2):204–269, 1993.

[3] J.A. Bergstra and J.W. Klop. "Algebra of communicating processes". In J.W. de Bakker, M. Hazewinkel, and J.K. Lenstra, editors, *Mathematics and Computer Science*, CWI Monograph 1, pages 89–138. North-Holland, Amsterdam, 1986.

[4] G. Boudol. "Flow event structures and flow nets". In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes,* Proceedings LITP Spring School on Theoretical Computer Science, La Roche Posay, France, volume 469 of LNCS, pages 62–95. Springer-Verlag, 1990.

[5] N. Busi. "Raffinamento di azioni in linguaggi concorrenti". Master's thesis, Università di Bologna, 1993.

[6] Ph. Darondeau and P. Degano. "Refinement of actions in event structures and causal trees". *Theoretical Computer Science*, 118(1):21–48, 1993.

[7] P. Degano and R. Gorrieri. "A causal operational definition of action refinement". *Information and Computation*. To appear.

[8] W.J. Fokkink. "The tyft/tyxt format reduces to tree rules". In *Proc. Theoretical Aspects of Computer Science,* Sendai, Japan, volume 789 of LNCS, pages 440–453. Springer-Verlag, 1994.

[9] R.J. van Glabbeek. "The refinement theorem for ST-bisimulation semantics". In M. Broy and C.B. Jones, editors, Proceedings IFIP TC2 Working Conference on *Programming Concepts and Methods,* Sea of Gallilea, Israel, pages 27–52. North-Holland, 1990.

[10] R.J. van Glabbeek. "Full abstraction in structural operational semantics" (extended abstract). In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, Proceedings of the Third International Conference on *Algebraic Methodology and Software Technology (AMAST'93),* Twente, The Netherlands, June 1993, Workshops in Computing, pages 77–84. Springer-Verlag, 1993.

[11] R.J. van Glabbeek and U. Goltz. "Equivalence notions for concurrent systems and refinement of actions". In A. Kreczmar and G. Mirkowska, editors, Proceedings *Mathematical Foundations of Computer Science 1993 (MFCS)*, Porąbka-Kozubnik, Poland, volume 379 of LNCS, pages 237–248. Springer-Verlag, 1989.

[12] R.J. van Glabbeek and U. Goltz. "Refinement of actions in causality based models". In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, REX Workshop on *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, Mook, The Netherlands 1989, volume 430 of LNCS, pages 267–300. Springer-Verlag, 1990.

[13] R.J. van Glabbeek and F.W. Vaandrager. "Petri net models for algebraic theories of concurrency". In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings PARLE conference*, Eindhoven, *Vol. II (Parallel Languages)*, volume 259 of LNCS, pages 224–242. Springer-Verlag, 1987.

[14] R. Gorrieri and C. Laneve. "Split and ST bisimulation semantics". *Information and Computation*. To appear.

[15] R. Gorrieri and C. Laneve. "The limit of split$_n$ bisimulations for CCS agents". In A. Tarlecki, editor, Proceedings *Mathematical Foundations of Computer Science 1991 (MFCS)*, Poland, volume 520 of LNCS, pages 170–180. Springer-Verlag, 1991.

[16] J.F. Groote and F.W. Vaandrager. "Structured operational semantics and bisimulation as a congruence". *Information and Computation*, 100(2):202–260, 1992.

[17] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, Cambridge, Massachusetts, 1988.

[18] M. Hennessy. "A proof system for weak ST bisimulation over a finite process algebra". Technical report, Computer Science Department, University of Sussex, 1991.

[19] M. Hennessy and R. Milner. "Algebraic laws for nondeterminism and concurrency". *Journal of the ACM*, 32(1):137–161, 1985.

[20] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, 1989.

[21] M. Nielsen, G.D. Plotkin, and G. Winskel. "Petri nets, event structures and domains", part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[22] G.D. Plotkin. "A structural approach to operational semantics". Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[23] A. Rensink. *Models and Methods for Action Refinement*. PhD thesis, University of Twente, 1993.

[24] W. Vogler. "Bisimulation and action refinement". *Theoretical Computer Science*, 114(1):173–200, 1993.

[25] J.L.M. Vrancken. "The algebra of communicating processes with empty process". Report FVI 86-01, Dept. of Computer Science, University of Amsterdam, 1986.