Information and Computation **170**, 95–133 (2001)
doi:10.1006/inco.2001.2967, available online at http://www.idealibrary.com on IDEAL®

Vertical Implementation

Arend Rensink[1]

Faculty of Informatics, University of Twente, Postbus 217, NL-7500 AE Enschede, The Netherlands
E-mail: rensink@cs.utwente.nl

and

Roberto Gorrieri

Dipartimento di Scienze dell'Informazione, University of Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy
E-mail: gorrieri@cs.unibo.it

Received May 28, 1998; final manuscript received October 20, 2000

We investigate criteria to relate specifications and implementations belonging to conceptually different levels of abstraction. For this purpose, we introduce the generic concept of a *vertical implementation relation*, which is a family of binary relations indexed by a *refinement function* that maps abstract actions onto concrete processes and thus determines the basic connection between the abstraction levels. If the refinement function is the identity, the vertical implementation relation collapses to a standard (horizontal) implementation relation. As desiderata for vertical implementation relations we formulate a number of congruence-like proof rules (notably a structural rule for recursion) that offer a powerful, compositional proof technique for vertical implementation. As a candidate vertical implementation relation we propose *vertical bisimulation*. Vertical bisimulation is compatible with the standard interleaving semantics of process algebra; in fact, the corresponding horizontal relation is rooted weak bisimulation. We prove that vertical bisimulation satisfies the proof rules for vertical implementation, thus establishing the consistency of the rules. Moreover, we define a corresponding notion of *abstraction* that strengthens the intuition behind vertical bisimulation and also provides a decision algorithm for finite-state systems. Finally, we give a number of small examples to demonstrate the advantages of vertical implementation in general and vertical bisimulation in particular.

Key Words: abstraction level, action refinement, bisimulation, compositionality, process algebra, vertical implementation.

Contents.

1. INTRODUCTION

There is a long tradition in defining process refinement theories (see, e.g., [8, 11, 29]), essentially based on the idea that, given two processes S and I, I is an implementation of S if I is more deterministic (or equivalent) according to the chosen semantics. Still, both S and I belong conceptually to the same abstraction level, as the actions they perform belong to the same alphabet. For this reason, we call such implementation relations *horizontal*.

In the development of software components, however, it is quite often required to compare systems that realize essentially the same functionality but belong to conceptually different abstraction levels, where the change of the level is usually accompanied by a change in the alphabets of actions they perform. For

[1] Partially supported by the European HCM network EXPRESS (Expressiveness of Languages for Concurrency), while the author was employed at the University of Hildesheim.

0890-5401/01 $35.00

such components, we would like to develop *vertical* implementation relations that, given an abstract process S and a concrete process I, tell us if I is a possible implementation for the specification S. This problem is rather unexplored, with the exception of the work on action refinement in process algebra [1, 2, 12, 17, 35, 41, 42], which however is not satisfactory in some respects (to be discussed later). The main contribution of this paper is to single out some sensible criteria that any vertical implementation relation should satisfy. Furthermore, we introduce one specific instance of such a vertical relation, which we call vertical bisimulation.

The concept of a vertical implementation relation $\leq^r$ entails the following:

1. It is parametric with respect to a *refinement* function r that maps abstract actions of the specification to concrete processes, thus fixing the implementation of the basic building blocks of the abstract system.

2. It is flexible enough (i) to offer several possible implementations for any given specification and (ii) not to require that the ordering of abstract actions is tightly preserved at the level of their implementing processes. To be more explicit, consider the following example: if $S = a; b$ and $r(a) = a_1; a_2$, then we would like to accept as legal implementations both $a_1; a_2; b$ and $a_1; (a_2 \,|||\, b)$, where in the latter an ordering at the abstract level (between a and b) has been partially forgotten at the concrete level (as a_2 and b are in parallel).

3. It is simple enough to be defined on the standard interleaving models of classic labeled transition systems. This has the advantage of making it possible to reuse most existing techniques developed for interleaving semantics.

4. It is a generalization of existing (horizontal) implementation relations; i.e., if the refinement function is the identity, then the vertical implementation relation, $\leq^{id}$, should collapse to some horizontal relation. This has two consequences: (i) the theory of horizontal and vertical implementation can be integrated uniformly, and (ii) the number of possible vertical relations is not less than the number of horizontal relations, at least in principle.

5. It is deadlock-freedom-preserving: Typically, if the specification is deadlock-free, we would expect that also the implementation is so.

6. It comes equipped with a set of sound congruence-like proof rules. This gives rise to a powerful, compositional proof technique to verify whether a certain process I is an implementation for some specification S.

The work on action refinement in process algebra satisfies few of these requirements. For instance, the existing theories say that the implementation of a specification S is given by $r(S)$ (the *syntactic substitution* of concrete processes $r(a)$ for actions a in S) [1, 2, 21, 31] or by $[\![r]\!]([\![S]\!])$ (the *semantic substitution* of the semantics of concrete processes $r(a)$ for actions a in the semantics of S) [12, 18, 22, 35]. Hence, the basic assumption of these theories is that there is *only one* possible implementation for a given specification; in other words, the action refinement function is used as a *prescriptive tool* to specify the only way abstract actions are to be implemented. Consequences of this are the following:

- The refinement function can be used as an operator of the language, as it also defines a function on processes. Hence, it becomes immediately relevant to investigate the so-called *congruence problem*: find an equivalence relation such that, if two processes S_1 and S_2 are equivalent, then also $r(S_1)$ and $r(S_2)$ are equivalent. Dating back to [9], it is clear that it is then necessary to move to *noninterleaving* semantics: the parallel execution of actions a and b, denoted $a \,|||\, b$, is equivalent in interleaving semantics to their sequential simulation $a; b + b; a$; however, if we refine a to the sequence $a_1; a_2$, then we obtain $a_1; a_2 \,|||\, b$ and $a_1; a_2; b + b; a_1; a_2$, which are not equivalent at all, as only the former may offer the execution sequence $a_1; b; a_2$. In [14, 23, 42, 43] it is shown that the coarsest congruence for the operator of action refinement contained in standard interleaving semantics is the ST semantics [19], a notion of equivalence that, roughly, considers actions as nonatomic activities split into two consecutive phases (see also [6]).

- Because of the strong relation to the (syntactical or semantical) structure of the specification S, the implementation $r(S)$ is quite rigidly defined. One of the typical constraints is that the possible causal relation between two abstract actions is strictly preserved among *all* the actions of the two implementing processes. For instance, if $S = a; b$ and $r(a) = a_1; a_2$ as above, then the only possible implementation

is $r(S) = a_1; a_2$; b and not the alternative $a_1; (a_2 \,|||\, b)$ where the causal relation between a and b is partially forgotten. This can be a serious drawback in practice, as pointed out in [28]. Some work has been devoted to define less rigid forms of action refinement that do allow some overlapping [13, 27, 34, 39, 44]. Still, in all these approaches, given a specification and a refinement function there is always only one possible implementation.

By allowing *more than one* implementation for a given specification, we get that in our approach the congruence problem simply disappears: since one single specification may admit nonequivalent implementations, *a fortiori* two equivalent specifications need not to have equivalent implementations. Hence, there is no longer a need to move to noninterleaving semantics.

The paper is organized as follows. In Section 2, our investigation starts by introducing the language we use (a mixture of CCS and CSP, containing all the well-known operators), equipped with operational and behavioural semantics (the standard notion of rooted weak bisimulation equivalence, see [29]). Then we present the class of refinement functions we use, which are restricted in that they map abstract actions to nonempty (i.e., not immediately terminating) processes that cannot deadlock.

Section 3 introduces the set of desired proof rules we would expect any vertical implementation $\preceq^r$ to satisfy. We use $\sqsubseteq^r$ as the formal symbol in the proof rules for a vertical implementation relation parameterized by r, and $\sqsubseteq$ (without parameter) for the corresponding standard (horizontal) implementation relation. The rules can be divided into three main groups. The first group states the interplay between $\sqsubseteq^r$ and $\sqsubseteq$: when r is the identity function, $\sqsubseteq^{id}$ reduces to $\sqsubseteq$; moreover, $\sqsubseteq^r$ and $\sqsubseteq$ compose, meaning that if, e.g., $S \sqsubseteq S'$ and $S' \sqsubseteq^r I'$ and $I' \sqsubseteq I$, then $S \sqsubseteq^r I$. The second group defines a set of congruence-like properties; e.g., if $S_i \sqsubseteq^r I_i$ for $i = 1, 2$, then $S_1 + S_2 \sqsubseteq^r I_1 + I_2$. Some of the congruence rules have side-conditions on r, in particular on the nature of the alphabets of processes refining distinct abstract actions. These side-conditions are necessary in order to obtain an intuitively sound (e.g., deadlock-freedom-preserving) proof system; we give some examples to illustrate this. The third group consists of a single rule that relaxes precisely the causality preservation constraint discussed above, and thus is a typical example of a rule that surmounts the intrinsic limitations of the standard approach to action refinement.

An interesting consequence of the proof system is that the (almost) standard notion of "action refinement as syntactic substitution" is a sound implementation technique (but by no means the only one) for any vertical relation that enjoys the proof rules.

Section 4 introduces vertical bisimulation, $\lesssim^r$, as the concrete notion of vertical implementation we propose in this paper. Its main features are the following:

- The underlying horizontal implementation relation is rooted weak bisimulation equivalence.
- Vertical bisimulation is formed of three components: a *down-simulation* (each abstract move must be matched by a sequence in the implementation), an *up-simulation* (each move of the implementation should find a justification either as the initial action of a *new* refined action or as a continuation of a *pending* refinement), and a *residual simulation*, requiring that each move of the pending refinements must be present in the implementation.
- $\lesssim^r$ enjoys all the proof rules: if $S \sqsubseteq^r I$ then $S \lesssim^r I$. In particular, $\lesssim^r$ reduces to rooted weak bisimulation equivalence when no action is refined. (The problem of finding a *complete* set of proof rules for $\lesssim^r$ is outside the scope of this paper.)

The proof system makes it possible to prove nontrivial facts in a completely proof-theoretic way. Alternatively, one may show directly that $\lesssim^r$ holds between two given (finite-state) systems, by providing an actual vertical bisimulation relation over their states. Furthermore, in Section 5 we also report another model-theoretic approach to check $\lesssim^r$ over finite-state transition systems: we introduce the *abstraction theorem*, according to which (under some constraints) a concrete transition system can be mapped algorithmically to a corresponding abstract transition system. Checking vertical bisimulation is then equivalent to first mapping the concrete-labeled transition system to its corresponding abstract one, and then checking (classical) rooted weak bisimulation between the two abstract transition systems.

Section 6 extends the result presented so far to the case of open terms. In particular, we discuss the proof rule of *recursion congruence*, which makes it possible to deduce vertical implementations of recursive, possibly infinite-state systems. Under the assumption of strict guardedness, recursion congruence is proved to hold for $\lesssim^r$.

Section 7 shows several variations of vertical implementation on an example taken from [7], as well as an example of a finite-state specification admitting an infinite-state implementation. Especially the latter demonstrates the fact that the proof rules can be used to deduce nontrivial vertical implementations. Finally, in Section 8 we discuss further extensions of our work, concerning possible variations of the notion of vertical implementation relation.

The proofs of this paper's most interesting theorems can be found in the Appendix; all proofs are given in the full report version [38]. A preliminary version of this paper appeared as [37].

2. BASIC DEFINITIONS

2.1. *The Language*

We assume a universe of action names $\mathbf{U}$, ranged over by a, b, c, an invisible action $\tau \notin \mathbf{U}$, and a termination label $\checkmark \notin \mathbf{U} \cup \{\tau\}$. Subsets of $\mathbf{U}$ are denoted $\mathbf{A}$, $\mathbf{C}$, A, C (where we sometimes use $\mathbf{A}$, A for *abstract* and $\mathbf{C}$, C for *concrete* actions, respectively). We denote $A_\tau = A \cup \{\tau\}$, $A_\checkmark = A \cup \{\checkmark\}$, and $A_{\tau,\checkmark} = A \cup \{\tau, \checkmark\}$ for any $A \subseteq \mathbf{U}$. $\mathbf{U}_{\tau,\checkmark}$ is ranged over by α, β, γ. Furthermore, we consider a universe of process variables, $\mathbf{X}$, ranged over by x, y, z; subsets of $\mathbf{X}$ are denoted X, Y. We define a family of languages $\mathbb{L}_\mathbf{A}$, indexed by the set of actions $\mathbf{A} \subseteq \mathbf{U}$ that may be used within terms and ranged over by t, u, v, according to the grammar

$$t ::= \mathbf{0} \mid \mathbf{1} \mid \alpha \mid t + t \mid t;\ t \mid t \parallel_A t \mid t[\phi] \mid t/A \mid x \mid \mu x.\, t,$$

where $\alpha \in \mathbf{A}_\tau$ (hence $\checkmark$ is not allowed in the language), $A \subseteq \mathbf{A}$, $\phi\colon \mathbf{A} \to \mathbf{A}$ and $x \in \mathbf{X}$. We drop the index $\mathbf{A}$ if it equals $\mathbf{U}$. The operators have the following intuitive meaning:

- $\mathbf{0}$ is a process that immediately deadlocks.
- $\mathbf{1}$ is a process that immediately terminates with a transition labeled $\checkmark$.
- α indicates the execution of the action α.
- $t + u$ indicates a choice between the behaviors described by the subterms t and u. The choice is decided by the first action (even if it is a $\checkmark$) that occurs from either subterm, after which the other subterm is discarded.
- $t; u$ is the sequential composition of t and u; i.e., t proceeds until it terminates, after which u takes over.
- $t \parallel_A u$ is the parallel composition of the behaviors described by t and u; A is a set of actions over which t and u synchronize. That is to say, actions from A can only be performed by both subterms in concert, whereas all other actions can be done by either subterm in isolation. In addition, we use the following special case of parallel composition:

$$t \parallel\!\parallel u = t \parallel_\emptyset u.$$

- $t[\phi]$ behaves as t, except that actions are renamed according to the function ϕ, extended when necessary with the mappings $\tau \mapsto \tau$ and $\checkmark \mapsto \checkmark$.
- t/A behaves as t, except that the actions in A are *hidden*, i.e., turned into internal actions.
- $x \in \mathbf{X}$ is a process variable, presumably bound by some encompassing recursive operator (see next item), or to be replaced by *syntactic substitution*: $t\langle u/x\rangle$ denotes the replacement within the term t of every (free) occurrence of x by the term u (see below for the formal definition).
- $\mu x.\, t$ with $x \in \mathbf{X}$ is a recursive term. It can be understood through its unfolding, $t\langle \mu x.\, t/x\rangle$. The variable x is considered to be *bound* in $\mu x.\, t$, meaning that it cannot be not affected by substitution. Therefore, the identity of bound variables is considered irrelevant; in fact, we apply the standard technique of identifying all terms up to renaming of the bound variables, meaning that if y is a fresh variable not occurring in t, then $\mu x.\, t$ and $\mu y.\, t\langle y/x\rangle$ are identified in all contexts.

We only consider recursion over *guarded* terms; see Definition 2.1 below.

TABLE 1

Free Variables and Syntactic Substitution

t	$\mathrm{fv}(t)$	$t\langle f\rangle$
$\mathbf{0}$	$\emptyset$	$\mathbf{0}$
$\mathbf{1}$	$\emptyset$	$\mathbf{1}$
α	$\emptyset$	α
$t_1 + t_2$	$\mathrm{fv}(t_1, t_2)$	$t_1\langle f\rangle + t_2\langle f\rangle$
$t_1; t_2$	$\mathrm{fv}(t_1, t_2)$	$t_1\langle f\rangle; t_2\langle f\rangle$
$t_1 \parallel_A t_2$	$\mathrm{fv}(t_1, t_2)$	$t_1\langle f\rangle \parallel_A t_2\langle f\rangle$
$t_1[\phi]$	$\mathrm{fv}(t_1)$	$t_1\langle f\rangle[\phi]$
t_1/A	$\mathrm{fv}(t_1)$	$t_1\langle f\rangle/A$
x	$\{x\}$	$\begin{cases} f(x) & \text{if } x \in \mathrm{dom}(f) \\ x & \text{otherwise} \end{cases}$
$\mu x.\, t_1$	$\mathrm{fv}(t_1)\backslash\{x\}$	$\mu y.\, (t_1\langle y/x\rangle\langle f\rangle)$ where $y \notin \mathrm{fv}(t) \cup \mathrm{dom}(f) \cup \mathrm{fv}(f)$

To formalize the notion of syntactic substitution, we first define the *free variables* of a term t, denoted $\mathrm{fv}(t)$, as those variables that do not occur in the scope of a recursion operator; see Table 1. We write $\mathrm{fv}(t, u) = \mathrm{fv}(t) \cup \mathrm{fv}(u)$. If $\mathrm{fv}(t) = \emptyset$ for a given term $t \in \mathbb{L}$, we call t *closed*; in contrast, we sometimes call a term *open* if it is not (known to be) closed. We will use $\mathbf{L_A}$ to denote the set of closed terms.

A *substitution function* f is a partial function from $\mathbf{X}$ to $\mathbb{L}_\mathbf{A}$; its domain of definition will be denoted $\mathrm{dom}(f)$. We use $\mathbf{X} \rightharpoonup \mathbb{L}_\mathbf{A}$ to denote the space of substitution functions; if f is a substitution function with $\mathrm{dom}(f) = X$, we write $f : \mathbf{X} \rightharpoonup \mathbb{L}_\mathbf{A}$ or $f : X \to \mathbb{L}_\mathbf{A}$. Table 1 defines the application of a substitution function f to a term t, denoted $t\langle f\rangle$, as the simultaneous replacement, within t, of every free occurrence of every variable $x \in \mathrm{dom}(f)$ by its image $f(x)$. Note the (standard) definition of substitution for recursive terms in Table 1: the bound variable is renamed to a variable not occurring in the term to be substituted, in order to avoid the capture of free variables.

Notation for Substitutions. If $\mathrm{dom}(f) = \{x_1, \ldots, x_n\}$ and f maps each x_i to a term t_i, we also write f as an explicit list of substitutions: $f = (t_1/x_1, \ldots, t_n/x_n)$ or $(t_i/x_i)_{1 \le i \le n}$. Correspondingly, we write $t\langle f\rangle = t\langle t_1/x_1, \ldots t_n/x_n\rangle$ or $t\langle t_i/x_i\rangle_{1 \le i \le n}$. The notion of free variables is extended to substitution functions by defining $\mathrm{fv}(f) = \bigcup_{x \in \mathrm{dom}(f)} \mathrm{fv}(f(x))$. As with terms, f is called closed if $\mathrm{fv}(f) = \emptyset$.

Guardedness. We can now also formalize the notion of *guardedness*, already mentioned above. This is a syntactic property: in principle, a variable $x \in \mathbf{X}$ is said to be guarded in a term $t \in \mathbb{L}$ if every occurrence of x is within a subterm $\alpha; u$ of t. The precise definition is slightly more flexible than that: for instance, x (but not y) is also considered to be guarded in $(y; a); x$. Furthermore, t is called *well guarded* if it contains only recursion over terms in which the recursion variable is guarded. Note that this does *not* imply that all variables are guarded in t; for instance, $t = x$ is well guarded but x is not guarded in t.

DEFINITION 2.1. First we define when an arbitrary term (in $\mathbb{L}$) is called *a guard*:

- $\mathbf{0}$ and α are guards (for all $\alpha \in \mathbf{A}_\tau$);
- $t + u$ is a guard if both t and u are guards;
- $t; u$ and $t \parallel_A u$ are guards if either t or u is a guard;
- t/A, $t[\phi]$, and $\mu x.\, t$ are guards if t is a guard;
- $\mathbf{1}$ and x are *not* guards (for all $x \in \mathbf{X}$).

Next we define when a variable is called *guarded* in a term:

- x is guarded in $\mathbf{0}$, $\mathbf{1}$ and α (for all $\alpha \in \mathbf{A}_\tau$);
- x is guarded in $t + u$ and $t \parallel_A u$ if x is guarded in both t and u;
- x is guarded in $t; u$ if x is guarded in t and either t is a guard or x is guarded in u;

- x is guarded in t/A, $t[\phi]$, and $\mu x.t$ if x is guarded in t;
- x is guarded in y ($\in \mathbf{X}$) if $x \neq y$.

We call $t \in \mathbb{L}$ *well guarded* if in all subterms $\mu x.u$ of t, x is guarded in u.

The following proposition states that both guardedness of a variable in a term and well-guardedness of a term are preserved by syntactic substitution.

PROPOSITION 2.1. *Let $t, u \in \mathbb{L}$ and $x, y \in \mathbf{X}$.*

1. *If x is guarded in t and u, then x is guarded in $t\langle u/y\rangle$.*
2. *If t and u are well guarded, then so is $t\langle u/y\rangle$.*

The set of well-guarded terms will be denoted $\mathbb{L}_{\mathbf{A}}^{\mathsf{wg}}$ (and accordingly $\mathbf{L}_{\mathbf{A}}^{\mathsf{wg}}$ for the closed fragment); however, where this does not give rise to confusion we will drop the superscript $^{\mathsf{wg}}$ and simply assume all terms to be implicitly well guarded.

2.2. *Operational Semantics*

The operational semantics is given in terms of labeled transition systems (LTSs). An LTS is a triple $\langle \Lambda, S, \rightarrow \rangle$ where Λ is a set of labels, S is a set of states, and $\rightarrow \subseteq S \times \Lambda \times S$ is the so-called transition relation. As usual, we denote $s \xrightarrow{\alpha} s'$ for $(s, \alpha, s') \in \rightarrow$, $s \xrightarrow{\alpha}$ for $\exists s'.\, s \xrightarrow{\alpha} s'$, and $s \not\xrightarrow{\alpha}$ for $\neg(s \xrightarrow{\alpha})$. If $\checkmark \in \Lambda$, this plays the special role of modelling *termination*. Accordingly, in any LTS we impose as a requirement on termination transitions (i.e., $\checkmark$-labeled ones) that they must lead to deadlocked states:

$$s \xrightarrow{\checkmark} s' \Rightarrow \nexists \alpha \in \Lambda.\, s' \xrightarrow{\alpha}. \tag{1}$$

The LTS for $\mathbf{L}_{\mathbf{A}}$ is obtained as follows. The set of labels is $\mathbf{A}_{\tau,\checkmark}$. The set of states is given by the terms of $\mathbf{L}_{\mathbf{A}}$. The transition relation is defined as the least relation satisfying the rules of Table 2. Note that these rules apply to open as well as closed terms; for instance, $a; x + y \xrightarrow{a} x$ is a derivable transition.

The termination label $\checkmark$ is needed to give a satisfactory semantic treatment of sequential composition (see [4]). Note that a choice may terminate even if only one of the operands terminates (cf. [5]). Finally, $\checkmark$ cannot be hidden and must be synchronized upon.

The following proposition shows the consequences of (well-)guardedness for the operational semantics: guarded variables are not "used" in initial transitions of a term, well-guardedness is preserved by transitions, and for well-guarded terms, the derivation rule for recursion can be replaced by another one in which the premise is structurally simpler than the conclusion (see also [3]). The latter fact has the consequence that more properties can be proved by induction on the term structure.

TABLE 2

Transition Rules

$$\frac{}{\mathbf{1} \xrightarrow{\checkmark} \mathbf{0}} \qquad \frac{}{\alpha \xrightarrow{\alpha} \mathbf{1}} \qquad \frac{t \xrightarrow{\alpha} t'}{t + u \xrightarrow{\alpha} t'} \qquad \frac{u \xrightarrow{\alpha} u'}{t + u \xrightarrow{\alpha} u'}$$

$$\frac{t \xrightarrow{\alpha} t' \quad \alpha \neq \checkmark}{t; u \xrightarrow{\alpha} t'; u} \qquad \frac{t \xrightarrow{\checkmark} t' \quad u \xrightarrow{\alpha} u'}{t; u \xrightarrow{\alpha} u'} \qquad \frac{t \xrightarrow{\alpha} t'}{t[\phi] \xrightarrow{\phi(\alpha)} t'[\phi]}$$

$$\frac{t \xrightarrow{\alpha} t' \quad \alpha \notin A_{\checkmark}}{t \parallel_A u \xrightarrow{\alpha} t' \parallel_A u} \qquad \frac{u \xrightarrow{\alpha} u' \quad \alpha \notin A_{\checkmark}}{t \parallel_A u \xrightarrow{\alpha} t \parallel_A u'} \qquad \frac{t \xrightarrow{\alpha} t' \quad u \xrightarrow{\alpha} u' \quad \alpha \in A_{\checkmark}}{t \parallel_A u \xrightarrow{\alpha} t' \parallel_A u'}$$

$$\frac{t \xrightarrow{\alpha} t' \quad \alpha \notin A}{t/A \xrightarrow{\alpha} t'/A} \qquad \frac{t \xrightarrow{\alpha} t' \quad \alpha \in A}{t/A \xrightarrow{\tau} t'/A} \qquad \frac{t\langle \mu x.t/x \rangle \xrightarrow{\alpha} t'}{\mu x.t \xrightarrow{\alpha} t'}$$

PROPOSITION 2.2. *Let $t, u \in \mathbb{L}$ and $x \in \mathbf{X}$.*

1. *If x is guarded in t and $t\langle u/x\rangle \xrightarrow{\alpha} t'$, then $t \xrightarrow{\alpha} t''$ for some t'' such that $t' = t''\langle u/x\rangle$.*
2. *If t is well guarded and $t \xrightarrow{\alpha} t'$, then t' is well guarded.*
3. *If t is well guarded, then the set of derivable transitions $t \xrightarrow{\alpha} t'$ remains the same if we replace the last rule of Table 2 by*

$$\frac{t \xrightarrow{\alpha} t'}{\mu x.\, t \xrightarrow{\alpha} t'\langle \mu x.\, t/x\rangle}.$$

The following property expresses that the semantics satisfies the condition on $\surd$-transitions imposed on an LTS.

PROPOSITION 2.3. *For all $\mathbf{A} \subseteq \mathbf{U}$, $\langle \mathbf{A}_{\surd,\tau}, \mathbf{L}_{\mathbf{A}}^{\mathsf{wg}}, \rightarrow\rangle$ is an LTS.*

2.3. *Behavioral Semantics*

In an interleaving operational semantics such as the above, a widely accepted τ-abstracting equivalence relation is *rooted (weak) bisimilarity*; see [29]. The definition is as follows. First, the basic, one-step transitions are extended to τ-abstracting transitions in the usual fashion: if $\sigma = \alpha_1 \cdots \alpha_n \in \mathbf{U}_{\tau,\surd}^*$ then

$$t \stackrel{\sigma}{\Rightarrow} u :\Leftrightarrow t \xrightarrow{\tau}{}^{*} \xrightarrow{\alpha_1} \xrightarrow{\tau}{}^{*} \cdots \xrightarrow{\tau}{}^{*} \xrightarrow{\alpha_n} \xrightarrow{\tau}{}^{*} u.$$

Furthermore, the definition relies on a function $\hat{\cdot}\colon \mathbf{U}_{\tau,\surd} \to \mathbf{U}_{\surd}^*$ such that $\hat{\tau} = \varepsilon$ (the empty string, with $t \stackrel{\varepsilon}{\Rightarrow} t$ for any t) and $\hat{\alpha} = \alpha$ for all $\alpha \in \mathbf{U}_{\surd}$.

DEFINITION 2.2. Let $T = \langle \Lambda, S, \rightarrow\rangle$ be a transition system.

- A *weak simulation* over T is a relation $\rho \subseteq S \times S$ such that for all $s_1\, \rho\, s_2$:
  —if $s_1 \xrightarrow{\alpha} s_1'$ then $\exists s_2 \stackrel{\hat{\alpha}}{\Rightarrow} s_2'$ such that $s_1'\, \rho\, s_2'$.

ρ is a weak *bi*simulation if ρ and ρ^{-1} are weak simulations.

- A *root* of a relation $\rho \subseteq S \times S$ is a subrelation $\tilde{\rho} \subseteq \rho$ such that for all $s_1\, \tilde{\rho}\, s_2$:
  —if $s_1 \xrightarrow{\tau} s_1'$ then $\exists s_2 \stackrel{\tau}{\Rightarrow} s_2'$ such that $s_1'\, \rho\, s_2'$.

$\tilde{\rho}$ is a *bi*root of ρ if $\tilde{\rho}$ is a root of ρ and $\tilde{\rho}^{-1}$ is a root of ρ^{-1}.

- *Weak bisimilarity* over T, denoted $\approx$, is the largest weak bisimulation over T, and *rooted bisimilarity*, denoted $\simeq$, is the largest biroot of $\approx$.

Note that, because we will use the same "rootedness" condition several times, we have defined it in a generic fashion. The following few examples illustrate the role played by termination:

$$\alpha \|_\alpha \mathbf{1} \simeq \mathbf{0} \not\approx \mathbf{1} \simeq \mathbf{1}+\mathbf{1} \not\approx \mathbf{1}+\alpha \not\approx \alpha.$$

The following result is well known (cf. [5, 29]):

PROPOSITION 2.4. *$\simeq$ is a congruence over $\mathbf{L}$. In particular, if $t \in \mathbb{L}$ and $f, g\colon \mathrm{fv}(t) \to \mathbf{L}$ such that $f(x) \simeq g(x)$ for all $x \in \mathrm{fv}(t)$, then $t\langle f\rangle \simeq t\langle g\rangle$.*

Restriction to Closed Terms. In the following sections, up to (but not including) Section 6, we are only considering bisimilarity of *closed* terms. We discuss the extension of bisimilarity to open terms in Section 6.

2.4. *Refinement Functions*

A *refinement function* maps abstract actions to concrete processes, where the notions of *abstract* and *concrete* are accompanied by a change of alphabet. For the purpose of this paper, in order to avoid

unnecessary complications, we single out the $\mathbf{L_A}$-fragment $\mathbf{R_A}$ of *refinement terms* that can be used as the refinement of abstract actions. $\mathbf{R_A}$ is generated by the following grammar

$$t ::= a \mid t + t \mid t;t,$$

where $a \in \mathbf{A}$. Then, if $\mathbf{A}$ is the set of abstract actions and $\mathbf{C}$ that of concrete actions ($\mathbf{A}$ and $\mathbf{C}$ not necessarily disjoint), a refinement function is of the form $r: \mathbf{A} \to \mathbf{R_C}$, with $\mathrm{dom}(r) = \mathbf{A}$, with the property that $r(a) \neq a$ for only a finite number of a (i.e., if $\mathbf{A}$ is infinite, then r is the identity almost everywhere).

The restriction of refinement images to $\mathbf{R}$ (subscript omitted when clear from context) is largely technically motivated. At the same time, however, the terms of $\mathbf{R}$ satisfy a number of intuitive sensibility criteria. Indeed, if a process t is to refine atomic action, then it is reasonable to require that t should be:

- *nonempty*, i.e., $t \not\xrightarrow{\surd}$. This comes down to the principle that a visible abstract activity a (i.e., something the environment can synchronize on) cannot simply disappear during refinement.
- *eventually terminating*, i.e., after a finite number of steps, t must terminate. This criterion essentially requires that the refinement of a given action cannot "get stuck" during execution. This seems quite reasonable in the light of the atomicity of the original (abstract) action.

Altogether, $\mathbf{R}$ is large enough to express sensible examples.

Confusion in Refinement Functions. In some circumstances, it is important that the refinements of distinct abstract actions themselves satisfy some distinctness criteria. The heart of the issue is *confusion* within the concrete system about the "origin" of actions.

EXAMPLE 2.1. Consider a refinement function with $a \mapsto c;\ a'$ and $b \mapsto c;\ b'$. On the abstract level, the actions a and b are distinct and cannot be confused. For instance, in $t = (a + d) \parallel_{a,b} b$ it is not possible that a and b synchronize; hence $t \simeq d;\ \mathbf{0}$. On the concrete level, however, the proposed refinements of a and b start with the same action; hence if a c occurs in the implementation, it is not a priori clear whether this reflects an abstract a or an abstract b. This may be especially problematic if parts of the concrete system synchronize over an occurrence of c, while one part intends the c to reflect an abstract a-occurrence, whereas the other part intends it to reflect an abstract b-occurrence. As an example, consider $u = (c; a' + d) \parallel_{a',b',c} c; b'$ (this being the direct, syntactic refinement of t above). One possible run is $u \xrightarrow{c} a' \parallel_{a',b',c} b' \simeq \mathbf{0}$, resulting in a deadlocked state; this run does not have a counterpart in t.

To avoid this and other types of confusion, we sometimes impose further restrictions on the refinement functions considered. To formulate these, first we define the *alphabet* of a refinement term $t \in \mathbf{R}$, which equals the set of actions that may be executed by t during its lifetime;

$$\mathcal{A}(a) = \{a\} \quad \mathcal{A}(t + u) = \mathcal{A}(t; u) = \mathcal{A}(t) \cup \mathcal{A}(u).$$

The intention of the alphabet is captured by the following (obvious) property:

PROPOSITION 2.5. *For all* $t \in \mathbf{R}$, $\mathcal{A}(t) = \{a \mid \exists \sigma: t \stackrel{\sigma a}{\Longrightarrow}\}$.

Using the alphabet, we now define two properties of refinement functions that rule out confusion of the kind exemplified above, to different degrees.

DEFINITION 2.3. Let $r: \mathbf{A} \to \mathbf{R_C}$ and $A \subseteq \mathbf{A}$ be given.

- r is said to *preserve* A if $\mathcal{A}(r(a)) \cap \mathcal{A}(r(b)) = \emptyset$ for all $a \in A$ and $b \in \mathbf{A} \backslash A$;
- r is said to be *distinct* on A if the following conditions hold:
  1. for all distinct $a \in A$ and $b \in \mathrm{dom}(r)$, $\mathcal{A}(r(a)) \cap \mathcal{A}(r(b)) = \emptyset$;
  2. for all $a \in A$ and all subterms $t + u$ and $t; u$ of $r(a)$, $\mathcal{A}(t) \cap \mathcal{A}(u) = \emptyset$.

r is simply called *distinct* if it is distinct on $\mathbf{A}$.

In other words, a refinement function r preserves a certain set $A \subseteq \mathrm{dom}(r)$ if there is no overlap between the actions occurring in the refinements of (the elements of) A and of (the elements of) $\mathrm{dom}(r)\backslash A$. For instance, the function r in Example 2.1 does not preserve $\{a\}$ or $\{b\}$, but it does preserve $\{a, b\}$ if a', b', and c do not occur in $r(d)$ for any $d \in \mathrm{dom}(r)\backslash\{a, b\}$. On the other hand, r is distinct on A if also the refinements of different actions in A have disjoint alphabets, and the images of individual actions in A contain no more than a single instance of any action. Hence distinctness implies preservation, and r in Example 2.1 is not distinct on $\{a\}$, $\{b\}$ or $\{a, b\}$.

Active Domain and Active Range. In the following it will be useful to distinguish the *active domain* $\mathrm{adom}(r)$ of a refinement function r, as well as its *active range* $\mathrm{arng}(r)$, defined as

$$\mathrm{arng}(r) = \bigcup_{r(a)\neq a} \mathcal{A}(r(a))$$

$$\mathrm{adom}(r) = \{a \mid r(a) \neq a\} \cup (\mathrm{arng}(r) \cap \mathrm{dom}(r)).$$

Hence the active domain is a subset of the domain, consisting of two types of actions: those that are not mapped onto themselves, and those that are used in the image of any action different from themselves. Note that $\mathrm{adom}(r)$ is always finite, due to the fact that we required r to be the identity almost everywhere (see above) and the fact that $\mathcal{A}(t)$ is a finite set for all $t \in \mathbf{R}$. It is interesting to observe that

$$\mathrm{arng}(r) = \mathcal{A}(r(\mathrm{adom}(r))),$$

hence justifying the name of active range.

EXAMPLE 2.2. If $\mathbf{A} = \mathbf{C} = \{a, b, c\}$ and $r: a \mapsto a; b,\ b \mapsto b,\ c \mapsto c$ then $\mathrm{adom}(r) = \{a, b\}$. Note that $\mathrm{adom}(r)$ is preserved by r.

The preservation of the active domain is a general property, formulated in the following proposition; in fact, this property is the reason why we introduced the active domain.

PROPOSITION 2.6. $\mathrm{adom}(r)$ *is the smallest* r*-preserved set containing* $\{a \mid r(a) \neq a\}$.

Refinement Function Constants and Operations. We use $id_{\mathbf{A}}: \mathbf{A} \to \mathbf{R_A}$ to denote the identity refinement function on $\mathbf{A}$ (hence $\mathrm{adom}(id_{\mathbf{A}}) = \emptyset$), omitting the index $\mathbf{A}$ if it is clear from the context. In addition, we use the following construction on refinement functions:

$$r\backslash A: a \mapsto \begin{cases} a & \text{if } a \in A \\ r(a) & \text{otherwise.} \end{cases}$$

Hence $r\backslash A$ turns r into the identity over the actions in A. Note that $r\backslash\mathrm{adom}(r) = id$ and if r preserves A then $\mathrm{adom}(r\backslash A) = \mathrm{adom}(r)\backslash A$.

3. PROOF RULES FOR VERTICAL IMPLEMENTATION

We now introduce the central concept of this paper, namely the notion of *vertical implementation*. When we write $t \preceq^r u$ we mean that t is an abstract system and u one of its possible implementations according to the (generic) *vertical implementation relation* $\preceq^r$, where the correspondence between actions of t and computations of u is set via the refinement function r. Similarly, when we write $t \preceq t'$ we mean that t and t' are two systems at the same abstraction level, related by the (generic) *horizontal* implementation relation $\preceq$ (i.e., relating systems at the same abstraction level), which could be one of those studied in, e.g., [15].

In this section we define a set of proof rules that any relation $\preceq^r$ should satisfy; in the proof rules, we use the syntactic symbol $\sqsubseteq^r$ for $\preceq^r$ and $\sqsubseteq$ for $\preceq$. The list of rules, expressing natural "desiderata," is reported in Table 3. Some of the proof rules have side conditions on their applicability, concerning

TABLE 3

Proof Rules for Vertical Implementation

$$\frac{}{t \sqsubseteq^{id} t}\mathrm{R}_1 \qquad \frac{t \sqsubseteq^{id} u}{t \sqsubseteq u}\mathrm{R}_2 \qquad \frac{t \sqsubseteq t' \quad t' \sqsubseteq^{r} u' \quad u' \sqsubseteq u}{t \sqsubseteq^{r} u}\mathrm{R}_3$$

$$\frac{}{\mathbf{0} \sqsubseteq^{r} \mathbf{0}}\mathrm{R}_4 \qquad \frac{}{\mathbf{1} \sqsubseteq^{r} \mathbf{1}}\mathrm{R}_5 \qquad \frac{}{\alpha \sqsubseteq^{r} r(\alpha)}\mathrm{R}_6 \qquad \frac{t_1 \sqsubseteq^{r} u_1 \quad t_2 \sqsubseteq^{r} u_2}{t_1 + t_2 \sqsubseteq^{r} u_1 + u_2}\mathrm{R}_7$$

$$\frac{t_1 \sqsubseteq^{r} u_1 \quad t_2 \sqsubseteq^{r} u_2}{t_1; t_2 \sqsubseteq^{r} u_1; u_2}\mathrm{R}_8 \qquad \frac{t \sqsubseteq^{r} u \quad \phi \upharpoonright \mathrm{adom}(r) = id_{\mathrm{adom}(r)}}{t[\phi] \sqsubseteq^{r} u[\phi]}\mathrm{R}_9$$

$$\frac{t \sqsubseteq^{r} u \quad r \text{ preserves } A}{t/A \sqsubseteq^{r \setminus A} u/\mathcal{A}(r(A))}\mathrm{R}_{10} \qquad \frac{t_1 \sqsubseteq^{r} u_1 \quad t_2 \sqsubseteq^{r} u_2 \quad r \text{ is distinct on } A}{t_1 \parallel_A t_2 \sqsubseteq^{r} u_1 \parallel_{\mathcal{A}(r(A))} u_2}\mathrm{R}_{11}$$

$$\frac{r(a) = u_1; u_2 \quad t \sqsubseteq^{r} v}{a; t \sqsubseteq^{r} u_1; (u_2 \lll v)}\mathrm{R}_{12}$$

distinctness and preservation on a set of actions by the refinement function r. To better understand them, below we present some examples showing that these side-conditions are really necessary.

Note that the rules in Table 3 range over closed terms only. The extension to open terms, including a structural rule for the congruence of the (second-order) recursion operator, is dealt with in the separate Section 6.

3.1. *Motivation*

Before illustrating the need for the side conditions, we discuss the rules of Table 3 in some detail. They can be divided in three groups.

- The first group of properties, consisting of rules R_1–R_3, expresses our basic assumption of compatibility of horizontal and vertical implementation. Rule R_1 simply states that every term implements itself as long as no proper refinement takes place, rule R_2 says that $\sqsubseteq^{id}$ implies $\sqsubseteq$, while Rule R_3 explains the interplay between horizontal and vertical implementation relations. Note that, as a consequence, we also have the derived rule

$$\frac{t \sqsubseteq u}{t \sqsubseteq^{id} u}$$

that, in conjunction with rule R_2, ensures that $\sqsubseteq$ and $\sqsubseteq^{id}$ are indeed the same relation.

Note also that Rules R_1 and R_2 imply that $\sqsubseteq$ is reflexive, whereas Rules R_1–R_3 together imply that $\sqsubseteq$ is transitive; hence $\sqsubseteq$ is a preorder, which indeed is the standard requirement for horizontal implementation relations.

Later in this paper, we choose weak bisimulation $\simeq$ to be the horizontal implementation relation $\leq$; however, the interplay between vertical implementation relations and horizontal implementation relations in no way should depend on this choice, and we feel that any of the τ-abstracting relations studied in, e.g., [15] can, in principle, be used as a basis.

- The rules R_4–R_{11} essentially express congruence of vertical implementation with respect to the operators of our language. For instance, if the refinement functions in these rules are set to *id*, then the properties expressed by these rules collapse to the standard precongruence properties of $\sqsubseteq$ for the operators of **L**. (In other words, the horizontal implementation relation $\sqsubseteq$ needs to be a precongruence, at least.)

Rules R_4 and R_5 simply express that deadlock and termination are independent of the abstraction level. Rule R_6 is the core of the relationship between the refinement function r and the vertical implementation relation. It expresses the basic expectation that $r(a)$ should be an implementation for a. Rules R_7, R_8, and R_{11} are quite obvious, as they inductively go into the structure of the components. Note that in Rule R_{11} the synchronization set A of the specification is refined in the implementation. We will comment below on its side condition.

R_9 can be used for renaming terms when the renaming function ϕ is an identity on actions to be refined. Observe that no proof rule for renaming is offered in case the renaming and refinement function interfere. There is some room for extension here; for instance, the following additional rule may be considered,

$$\frac{t \sqsubseteq^r u \quad \phi \text{ injective}}{t[\phi] \sqsubseteq^{\psi \circ r \circ \phi^{-1}} u[\psi]},$$

where $\psi \circ r \circ \phi^{-1}$ is a construction on the refinement function r with the obvious meaning. At this point, however, we have chosen not to be exhaustive but rather to concentrate on the essential rules.

Rule R_{10} is a similar congruence-like rule for hiding, with the proviso that the refinement function has lost some of its active domain in correspondence to those actions that are hidden. We will comment later on its side condition. An interesting consequence of R_{10} is given by the (derived) rule

$$\frac{t \sqsubseteq^r u}{t/\mathrm{adom}(r) \sqsubseteq u/\mathrm{arng}(r)}.$$

(Note that in this case, the side condition can be dropped, since r always preserves adom(r).) Hence, by hiding all the actions that are refined, the vertical implementation is turned back into a horizontal implementation relation.

- Rule R_{12} states that in certain circumstances, sequential composition in the specification need not be taken literally: there may be overlap between the tail of (the refinement of) the first operand and (the implementation of) the second operand. In other words, this rule expresses a certain degree of *weakening* of the causal ordering during refinement in the sense discussed in the Introduction, reminiscent of the approaches of [28] and [44].

Note that, because of our choice of "refinement language" **R**, it is clear that u_1 is not terminated in the premise $r(a) = u_1; u_2$. This is indeed a necessary circumstance. To see why, note that we do also *not* expect the following generalization of R_{12} to hold:

$$\frac{s \sqsubseteq^r u_1; u_2 \quad t \sqsubseteq^r v}{s; t \sqsubseteq^r u_1; (u_2 \,|||\, v)}.$$

If $a \equiv \mathbf{1}; a$ (where $\equiv = \sqsubseteq \cap \sqsupseteq$), this more general rule—which in fact also generalizes R_8—would allow one to derive $a; b \sqsubseteq^{id} a \,|||\, b$ and hence $a; b \sqsubseteq a \,|||\, b$; this is not consistent with a deadlock-preserving horizontal implementation relation ($a; b$ cannot deadlock when synchronized with $a; b + a; \mathbf{0}$ whereas $a \,|||\, b$ can).

The use of Rule R_{12} is fairly limited; in the conclusion of this paper we discuss another, more general rule for the weakening of sequential composition. Nevertheless, there are nontrivial applications even of this limited version, as we show in Section 7.

Two small illustrations of the rules are provided by the following examples, which were already discussed in the Introduction. The first one is based on the assumption that we are working in an interleaving model, where $a \,|||\, b \sqsubseteq a; b + b; a$.

EXAMPLE 3.1. Using Table 3, we can show that if a is split into $r(a) = a_1; a_2$ in the abstract system $S = a \,|||\, b$, this can be implemented either by treating (the implementation of) a and b as independent, resulting in $I_1 = a_1; a_2 \,|||\, b$,

$$\frac{\dfrac{}{a \sqsubseteq^r a_1; a_2}\mathrm{R}_6 \quad \dfrac{}{b \sqsubseteq^r b}\mathrm{R}_6}{S \sqsubseteq^r I_1}\mathrm{R}_{11},$$

or by imposing an ordering, as in $I_2 = a_1; a_2; b + b; a_1; a_2$,

$$\frac{\dfrac{\vdots}{S \sqsubseteq a; b + b; a} \quad \dfrac{\vdots}{a; b + b; a \sqsubseteq^r I_2}\mathrm{R}_8, \mathrm{R}_7 \quad \dfrac{\dfrac{}{I_2 \sqsubseteq^{id} I_2}\mathrm{R}_1}{I_2 \sqsubseteq I_2}\mathrm{R}_2}{S \sqsubseteq^r I_2}\mathrm{R}_3.$$

The second example concerns the weakening of the causality using R_{12}.

EXAMPLE 3.2. Consider again $r\colon a \mapsto a_1; a_2,\ b \mapsto b$, and let $S = a; b$ and $I = a_1; (a_2 \,|||\, b)$:

$$\frac{r(a) = a_1; a_2 \quad \overline{b \sqsubseteq^r b}\,\mathrm{R}_6}{S \sqsubseteq^r I}\,\mathrm{R}_{12}.$$

More interesting examples can be found in Section 7.

3.2. *Side Conditions*

Although the side conditions on rules R_{11} and R_{10} in Table 3 seem ugly, it is not difficult to see that they are necessary, at least if one wants to meet the minimal requirement of choosing a horizontal implementation relation that preserves deadlock freedom (i.e., such that if t is deadlock-free and $t \leq u$ then u is deadlock-free). We show a few examples illustrating some of the most striking problems. The first example concerns the side condition of Rule R_{10} for hiding.

EXAMPLE 3.3. Let $r\colon \mathbf{A} \to \mathbf{R_C}$ be a refinement function with active part $a \mapsto a; c$ and $b \mapsto b; c$ (where $c \notin \mathbf{A}$). Hence r preserves neither $\{a\}$ nor $\{b\}$. Then, if we use the proof rule ignoring the side conditions, we would get the derivation

$$\frac{\dfrac{\dfrac{}{a, \sqsubseteq^r a; c}\,\mathrm{R}_6}{a/b \sqsubseteq^{r\setminus b} (a; c)/b, c}\,\mathrm{R}_{10} \quad \dfrac{}{a \sqsubseteq^{r\setminus b} a; c}\,\mathrm{R}_6}{(a/b) \parallel_a a \sqsubseteq^{r\setminus b} ((a; c)/b, c) \parallel_{a,c} a; c}\,\mathrm{R}_{11}}{((a/b) \parallel_a a)/a \sqsubseteq (((a; c)/b, c) \parallel_{a,c} a; c)/a, c}\,\mathrm{R}_{10}, \mathrm{R}_2.$$

The left-hand term on the bottom line contains no deadlock (there is just one transition of synchronization, leading to the terminated state $((\mathbf{1}/b) \parallel_a \mathbf{1})/a$), whereas the right hand term has the transition

$$(((a; c)/b, c) \parallel_{a,c} a; c)/a, c \xrightarrow{\tau} ((\mathbf{1}; c/b, c) \parallel_{a,c} \mathbf{1}; c)/a, c$$

to a deadlocked state. This contradicts the requirement that $\leq$ preserves deadlock freedom.

The next few examples show problems that derive from unintended effects of Rule R_{11} for parallel composition if we omit its side condition of distinctness.

EXAMPLE 3.4. Let r be a refinement function with active part $a \mapsto c; b + c; d$. The rules of Table 3 then allow the derivation

$$\frac{\dfrac{\dfrac{}{a \sqsubseteq^r c; b + c; d}\,\mathrm{R}_6 \quad \dfrac{}{a \sqsubseteq^r c; b + c; d}\,\mathrm{R}_6}{a \parallel_a a \sqsubseteq^r (c; b + c; d) \parallel_{b,c,d} (c; b + c; d)}\,\mathrm{R}_{11}}{(a \parallel_a a)/a \sqsubseteq ((c; b + c; d) \parallel_{b,c,d} (c; b + c; d))/b, c, d}\,\mathrm{R}_{10}, \mathrm{R}_2.$$

The left-hand term on the bottom line contains no deadlock (there is just one transition of synchronization, leading to the terminated state $(\mathbf{1} \parallel_a \mathbf{1})/a$), whereas the right-hand term has the transition

$$((c; b + c; d) \parallel_{b,c,d} (c; b + c; d))/b, c, d \xrightarrow{\tau} (\mathbf{1}; b \parallel_{b,c,d} \mathbf{1}; d)/b, c, d$$

to a deadlocked state. This contradicts the requirement that $\leq$ preserves deadlock freedom.

It is clear that the problem is related to the fact that $r(a)$ is *nondeterministic* and that, in synchronization, local choices can show up inconsistent at a global level. The side condition of distinctness solves the problem by invalidating the above derivation, since r violates Condition 2 of distinctness (Definition 2.3). Much the same problem can also be generated if the refinements of two different abstract actions start with the same concrete action (hence violating condition 1 of Definition 2.3), as the following example shows.

EXAMPLE 3.5. Let r be a refinement function with active part $a \mapsto c;a$ and $b \mapsto c;b$. The rules of Table 3 then allow the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a \sqsubseteq^r c;a}\mathrm{R}_6 \quad \dfrac{}{d \sqsubseteq^r d}\mathrm{R}_6}{a+d \sqsubseteq^r c;a+d}\mathrm{R}_7 \quad \dfrac{\dfrac{}{b \sqsubseteq^r c;b}\mathrm{R}_6 \quad \dfrac{}{d \sqsubseteq^r d}\mathrm{R}_6}{b+d \sqsubseteq^r c;b+d}\mathrm{R}_7}{(a+d) \parallel_{a,b} (b+d) \sqsubseteq^r (c;a+d) \parallel_{a,b,c} (c;b+d)}\mathrm{R}_{11}}{((a+d) \parallel_{a,b} (b+d))/a,b \sqsubseteq ((c;a+d) \parallel_{a,b,c} (c;b+d))/a,b,c}\mathrm{R}_{10}, \mathrm{R}_2.$$

The left-hand term on the bottom line contains no deadlock (all transition sequences lead to a terminated state), whereas the right-hand term has the transition

$$((c;a+d) \parallel_{a,b,c} (c;b+d))/a,b,c \xrightarrow{\tau} (\mathbf{1};a \parallel_{a,b,c} \mathbf{1};b)/a,b,c$$

to a deadlocked state. This contradicts the requirement that $\leq$ preserves deadlock freedom.

To prevent this sort of things from occurring, it is necessary that distinct synchronizing abstract actions are refined to terms having disjoint initial actions. Moreover, initial actions and later (i.e., noninitial) actions of a refinement image $r(a)$ should also be kept distinct, as the following example shows.

EXAMPLE 3.6. Given a refinement function r that is the identity everywhere except for $a \mapsto c;c$, we have that $(a;b \parallel_a ((a+b) \between (a+b)))/a,b$ is implemented by $(c;c;b \parallel_c ((c;c+b) \between (c;c+b)))/c,b$. However, while the former cannot deadlock, the latter can.

Again, imposing the side condition of distinctness invalidates the derivation, since the refinement function r in this example violates Condition 2 of Definition 2.3.

The identification of suitable side conditions for rule R_{11} is related to the issue of "syntactic versus semantic action refinement" studied in [21], where suitable conditions for syntactic substitution to distribute over the parallel operator (with synchronization) have been singled out. Hence, different side conditions for different classes of refinable terms are possible, according to the results presented in that paper.

3.3. *Syntactic Refinement*

Although the derivation rules in Table 3 give no recipe for deriving implementations from specifications, one particular implementation can in many cases be obtained through the *syntactic substitution* of all abstract actions by their refinements. For a given refinement function $r: \mathbf{A} \to \mathbf{R_C}$, syntactic substitution can be formalized as a partial function $r^*: \mathbb{L}_{\mathbf{A}} \rightharpoonup \mathbb{L}_{\mathbf{C}}$, defined in Table 4. The partial definedness (originating in the rules for parallel composition, renaming and hiding) is necessary to ensure that syntactic refinement (if defined) always yields a valid $\sqsubseteq^r$-implementation. We then have the following result:

TABLE 4

Syntactic Refinement

$$\begin{aligned}
r^*(\mathbf{0}) &:= \mathbf{0} \\
r^*(\mathbf{1}) &:= \mathbf{1} \\
r^*(\alpha) &:= r(\alpha) \\
r^*(t+u) &:= r^*(t) + r^*(u) \\
r^*(t;u) &:= r^*(t);\, r^*(u) \\
r^*(t \parallel_A u) &:= r^*(t) \parallel_{\mathcal{A}(r(A))} r^*(u) \quad \text{if } r \text{ is distinct on } A \\
r^*(t[\phi]) &:= r^*(t)[\phi] \quad \text{if } \phi \restriction \mathrm{adom}(r) = id_{\mathrm{adom}(r)} \\
r^*(t/A) &:= r^*(t)/\mathcal{A}(r(A)) \quad \text{if } r \text{ preserves } A \\
r^*(x) &:= x \\
r^*(\mu x.\, t) &:= \mu x.\, r^*(t)
\end{aligned}$$

THEOREM 3.1. *For all recursion-free* $t \in \mathbf{L_A}$ *and* $r: \mathbf{A} \to \mathbf{R}$, *if* r^* *is defined on* t, *then* $t \sqsubseteq^r r^*(t)$.

Proof. By induction on the structure of t, thanks to the rules in Table 3. ■

Note that this result is limited, not just to closed terms, but to recursion-free terms, i.e., with finite behavior. The reason is that our current proof system does not allow reasoning about recursion. We will repair this omission in Section 6.

One could turn the fact that syntactic refinement implies vertical implementation around and *define* a vertical implementation relation in terms of syntactic action refinement, taking care to interpret the specification and implementation up to some horizontal implementation relation or equivalence such as, for instance, bisimulation.[2] This gives rise to

$$t \leq^r u \quad :\Leftrightarrow \quad \exists v: t \simeq v, r^*(v) \simeq u.$$

$\leq^r$ meets several of the requirements discussed in the Introduction; for instance, we have that $a \,|||\, b \leq^r a_1; a_2 \,|||\, b$ as well as $a \,|||\, b \leq^r a_1; a_2; b + b; a_1; a_2$ if $r: a \mapsto a_1; a_2$ (see also Example 3.1), showing that a single specification can have incomparable vertical implementations. In fact, $\leq^r$ satisfies all the proof rules of Table 3, with the exception of R_{12}.

A similar technique can be used to define a vertical implementation relation using *semantic* rather than syntactic refinement. This again gives rise to a relation that satisfies all proof rules but R_{12}. It appears that R_{12} is typical of the flexibility one would like to have in implementing causality but is excluded by the traditional approach to action refinement.

4. VERTICAL BISIMULATION

We now define an actual vertical implementation relation that satisfies all the proof rules of Table 3. This section starts by introducing the basic definition, built on rooted weak bisimulation (see Definition 2.2), chosen as the horizontal implementation relation. Then we present the main results of our vertical bisimulation relation, namely soundness of the rules in Table 3 (even if such a set of rules is not complete) and (as a consequence) soundness of syntactic refinement.

4.1. *The Relation*

As we have seen, rooted weak bisimulation is defined using bisimulation relations that connect states of the specification with states of the implementation and vice versa. In an analogous way, we define vertical bisimulation as the combination of unidirectional simulations. However, in contrast to weak bisimulation, the directions are no longer symmetric. To simulate the abstract transitions of the specification by the implementation, we define the concept of *down-simulation*, according to which abstract transitions are matched with complete runs of the corresponding refinements in the implementation.

In the following definitions, $T = \langle \mathbf{U}_{\tau,\surd}, S, \to \rangle$ is a fixed transition system, and $r: \mathbf{A} \to \mathbf{R_C}$ a refinement function.

DEFINITION 4.1. A *down-simulation up to* r over T is a binary relation $\rho \subseteq S \times S$ such that for all $s_1 \,\rho\, s_2$, if $s_1 \xrightarrow{\alpha} s_1'$ then one of the following holds:

1. $\alpha \in \mathrm{adom}(r)$, and if $r(\alpha) \stackrel{\sigma\surd}{\Longrightarrow}$ then $\exists s_2 \stackrel{\sigma}{\Rightarrow} s_2'$ such that $s_1' \,\rho\, s_2'$;
2. $\alpha \notin \mathrm{adom}(r)$, and $\exists s_2 \stackrel{\hat{\alpha}}{\Rightarrow} s_2'$ such that $s_1' \,\rho\, s_2'$

It follows that down-simulation is a rather weak notion: w.r.t. the implementation, it regards only complete runs of the refined actions. The intermediate states of the implementation, traversed during such a complete run, are not investigated at all.

[2] This observation is due to an anonymous referee.

EXAMPLE 4.1. Let $r: a \mapsto a_1; a_2, b \mapsto b, c \mapsto c$. There is a down-simulation between $S = a; b$ and $I = a_1; (a_2; b + c)$, given by $\{(S, I), (\mathbf{1}; b, \mathbf{1}; b), (\mathbf{1}, \mathbf{1}), (\mathbf{0}, \mathbf{0})\}$; this does not investigate the intermediate state $\mathbf{1}; (a_2; b + c)$ that the implementation passes through while doing $I \stackrel{a_1 a_2}{\Longrightarrow} \mathbf{1}; b$.

To define the dual notion of *up-simulation*, we also must take into account that in any given state of the implementation, there may be associated refined actions whose execution has not yet terminated. These will be collected in a set of *residual* (or *pending*) *refinements* that will be used to parameterize the bisimulation.

To be precise, an r-residual set will be a multiset of nonterminated proper derivatives of r-images. Such a set is formally represented by a function $R: \mathbf{L} \to \mathbb{N}$. We will denote $t \in R$ if $R(t) > 0$. To be precise, the collection of residual sets of r is defined as

$$rsd(r) = \left\{ R: \mathbf{L} \to \mathbb{N} \;\middle|\; \forall t \in R: \exists a \in \mathrm{dom}(r), \exists \sigma \in \mathbf{U}^+: r(a) \stackrel{\sigma}{\Rightarrow} t \not\stackrel{\checkmark}{\longrightarrow} \right\}.$$

(Note that we cannot require $R: \mathbf{R} \to \mathbb{N}$, even though r maps to $\mathbf{R}$ only, since the derivatives of terms in $\mathbf{R}$ may contain occurrences of $\mathbf{1}$.) We use the following constructions over residual sets:

$$\begin{aligned}
\emptyset\text{:}\quad & u \mapsto 0 \\
[t]\text{:}\quad & u \mapsto \begin{cases} 1 & \text{if } u = t \text{ and } t \not\stackrel{\checkmark}{\longrightarrow} \\ 0 & \text{otherwise} \end{cases} \\
R_1 \oplus R_2\text{:}\quad & u \mapsto R_1(u) + R_2(u) \\
R_1 \ominus R_2\text{:}\quad & u \mapsto max\{R_1(u) - R_2(u), 0\}.
\end{aligned}$$

The behavior of a residual set corresponds to the synchronization-free parallel composition of its elements. Formally,

$$R \stackrel{\alpha}{\to} R' :\Leftrightarrow \exists t \in R: \exists t \stackrel{\alpha}{\to} t': R' = (R \ominus [t]) \oplus [t'].$$

Note the fact that terminated terms do not contribute to the residual set. The reason we can ignore terminated terms is that it is certain that such terms no longer display any operational behavior.

An up-simulation must maintain the multiset of residual refinements: either the implementation's move corresponds to the initial concrete action of a refined abstract action, or it is an action of a residual refinement. The residual set forms an additional component to every pair of (specification and implementation) states; hence we have a *ternary* rather than a binary relation.[3]

Notation for Ternary Relations. In the following, we will often work with ternary relations of the form $\rho \subseteq S \times S \times rsd(r)$ for some set of states S and refinement function r. We use the notation $s_1 \, \rho^R s_2$ to abbreviate $(s_1, s_2, R) \in \rho$; in other words, ρ^R is interpreted as the binary relation $\{(s_1, s_2) \mid (s_1, s_2, R) \in \rho\}$.

DEFINITION 4.2. An *up-simulation up to* r over T is a ternary relation $\rho \subseteq S \times S \times rsd(r)$ such that for all $s_1 \, \rho^R \, s_2$, if $s_2 \stackrel{\gamma}{\to} s_2'$ then one of the following holds:

1. $\exists \alpha \in \mathrm{adom}(r): \exists s_1 \stackrel{\alpha}{\Rightarrow} s_1'$ and $\exists r(\alpha) \stackrel{\gamma}{\to} v$ such that $s_1' \, \rho^{R \oplus [v]} \, s_2'$;
2. $\exists s_1 \stackrel{\varepsilon}{\Rightarrow} s_1'$ and $\exists R \stackrel{\gamma}{\to} R'$ such that $s_1' \, \rho^{R'} \, s_2'$;
3. $\gamma \notin \mathrm{arng}(r)$ and $\exists s_1 \stackrel{\hat{\gamma}}{\Rightarrow} s_1'$ such that $s_1' \, \rho^R \, s_2'$.

Note that when the implementation move is matched by the pending refinements in the residual set (item 2), then the specification is allowed to move silently (i.e., with a τ-transition). See [38] for a discussion on variations on this definition, showing among other things that this is indeed necessary.

To combine a (binary) down-simulation ρ_1 and a (ternary) up-simulation ρ_2, we require that ρ_1 equals the subrelation $\rho_2^{\emptyset}$, i.e., where the residual set component is empty. (This is the natural choice, since in

[3] Other ternary bisimulation-based relations are, for instance, *history-preserving* bisimulation [16], and *symbolic* bisimulation [25].

the definition of down-simulation we assumed to investigate only states of the implementation where all refinements had been simulated completely.) Unfortunately, such a combination does not yet give rise to a useful notion of vertical implementation, since there is no guarantee that refinements that were started (and hence are in the residual set) can be finished.

EXAMPLE 4.2. Let $r: a \mapsto a_1; a_2$ and consider $S = a$ and $I = a_1; \mathbf{0} + a_1; a_2$. Down- and up-simulations between S and I are given by

$$\rho_1 = \{(S, I), (\mathbf{1}, \mathbf{1}), (\mathbf{0}, \mathbf{0})\}$$
$$\rho_2 = \{(S, I, \emptyset), (\mathbf{1}, \mathbf{1}; \mathbf{0}, [a_2]), (\mathbf{1}, \mathbf{1}; a_2, [a_2]), (\mathbf{1}, \mathbf{1}, \emptyset), (\mathbf{0}, \mathbf{0}, \emptyset)\}.$$

Note that $\rho_1 = \rho_2^{\emptyset}$. However, we certainly do not want I as an implementation of S, since I may be not able to complete the sequence implementing a, and may deadlock instead. In particular, we have $\mathbf{1}\, \rho^{[a_2]}\, \mathbf{1}; \mathbf{0}$, which relates a terminated state with a deadlocked state. In fact, Rules R_{10} and R_2 would allow us to derive $S/a \simeq I/a_1, a_2$ from $S \lesssim^r I$, which is false.

Vertical bisimulation, therefore, is determined by a relation that is both a down-simulation, an up-simulation, and a *residual simulation*; the latter requires that any move of the pending refinement set must be matched by the implementation, without the specification moving at all. This implies that pending refinements can be "worked off" in any possible order, or indeed in parallel, by the implementation. This property can be construed as an operational formulation of *atomicity*: that which is started can always be finished.

DEFINITION 4.3. Let r be a refinement function.

- A *weak vertical bisimulation up to r* over T is a ternary relation $\rho \subseteq S \times S \times rsd(r)$ such that
  1. $\rho^{\emptyset}$ is a down-simulation;
  2. ρ is an up-simulation;
  3. $\{(R, s_2) \mid s_1\, \rho^R s_2\}$ is a weak simulation (called *residual simulation*) for all $s_1 \in S$.
- *Weak vertical bisimilarity up to r* over T, denoted $\preccurlyeq^r$, is the largest weak vertical bisimulation up to r over T, and *rooted vertical bisimilarity up to r* over T, denoted $\lesssim^r$, is the largest biroot of $\preccurlyeq^{r,\emptyset}$.

Figure 1 shows an example of an actual vertical bisimulation: given $r: a \mapsto a_1; a_2$, the figure shows vertical bisimulations proving $a; b \lesssim^r a_1; a_2; b$ and $a; b \lesssim^r a_1; (a_2 \,|||\, b)$. The dotted lines connect related states and their labeling is the residual set indexing the relation.

4.2. *Soundness Results*

Directly from Definition 4.3, the following consistency result follows:

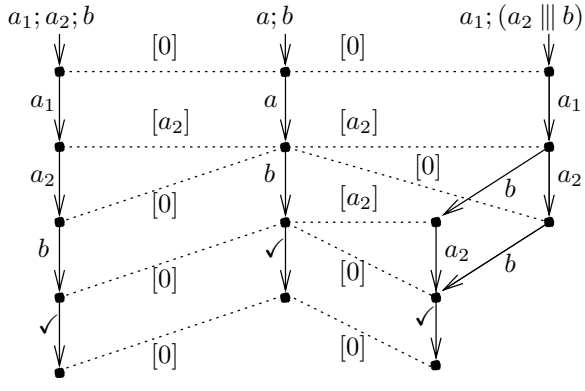


FIG. 1. An example of vertical bisimulation: with $r: a \mapsto a_1; a_2$.

PROPOSITION 4.1. *$\preccurlyeq^{id,\emptyset} = \approx$ and $\lesssim^{id} = \simeq$.*

(This easily follows by noting that $\mathrm{adom}(id) = \emptyset$; hence Clause 2 of Definition 4.1 and Clause 3 of Definition 4.2 always apply. Moreover, $rsd(id) = \{\emptyset\}$; hence there can be no proper pending refinements.) This immediately implies the soundness of Rules R_1 and R_2 for closed terms. In fact, we can prove soundness of each of the proof rules in Table 3—in particular also of R_{12}.

THEOREM 4.1. *$\lesssim^r$ satisfies all the rules in Table 3.*

The proof is deferred to the Appendix. The soundness result ensures that whenever $S \sqsubseteq^r I$ is provable in the proof system of Table 3, then $S \lesssim^r I$. As an immediate consequence, we get the following property (see Theorem 3.1):

COROLLARY 4.1. *For all recursion-free $t \in \mathbf{L_A}$ and $r : \mathbf{A} \to \mathbf{R_C}$, if r^* is defined on t, then $t \lesssim^r r^*(t)$.*

Another relevant property we want our vertical bisimulation to satisfy is the preservation of deadlock freedom: if $S \lesssim^r I$ and S is deadlock-free, then also I is deadlock-free. We first need to formally define when an agent is deadlock-free.

DEFINITION 4.4. A process $t \in \mathbf{L_A}$ is *deadlock-free* if for all $t' \in \mathbf{L_A}$ such that $\exists \sigma \in \mathbf{A}^* : t \stackrel{\sigma}{\Rightarrow} t'$, it follows that $\exists \alpha \in \mathbf{A}_{\tau,\surd} : t' \stackrel{\alpha}{\rightarrow}$.

We say that a binary relation preserves deadlock freedom if whenever the left-hand side is deadlock-free then so is the right-hand side. Directly from the definition of vertical bisimulation, it follows that

THEOREM 4.2. *$\lesssim^r$ preserves deadlock freedom.*

As a trivial corollary of Theorems 4.2 and 4.1, we have that also *provable* vertical implementation (using the proof rules of Section 3) preserves deadlock freedom.

COROLLARY 4.2. *$\sqsubseteq^r$ preserves deadlock freedom.*

5. ABSTRACTION

In order to strengthen the intuition behind vertical bisimulation, in this section we show that it can in fact be characterized as a combination of (horizontal) rooted weak bisimilarity and *abstraction*. Building the abstraction of a transition system U up to a given refinement function consists of "guessing" where the transitions of U originate from, i.e., which abstract actions they refine. The states of the abstraction of U are therefore pairs (s, R), where s is a state of U and R is the residual refinement set of those abstract actions that have been already guessed. The transitions of the abstraction of U are essentially the same as those of U, but with a different labeling: if a transition of U "opens" a new refinement, then the corresponding transition of the abstraction is labeled with the action being refined; if such transition "continues" a pending refinement, then (this must be matched by the residual set of the state of the abstraction and) the labeling is the invisible action τ. Then some constraints, called *saturation conditions*, which resemble the three simulation-like conditions of the definition of vertical bisimulation, are to be satisfied.

Note that, in this section, we consider transition systems with a further component $q_T \in S_T$, denoting the *initial state*. We furthermore consider (rooted) weak bisimilarity and vertical bisimilarity as holding between transitions systems T and U rather than within a single transition system; i.e., $T \approx U$, $T \simeq U$ or $T \lesssim^r U$. This is interpreted as meaning that the initial states of T and U are related (i.e., $q_T \approx q_U$ etc.) when regarded within the disjoint union of the transition systems, $T \uplus U$.

DEFINITION 5.1. Let U be a transition system with $\Lambda_U = \mathbf{C}_{\tau,\surd}$, and $r : \mathbf{A} \to \mathbf{R_C}$ a refinement function. An *r-abstraction of U* is a transition system $\langle \mathbf{A}_{\tau,\surd}, S, \rightarrow, (q_U, \emptyset) \rangle$, where $S \subseteq S_U \times rsd(r)$ and

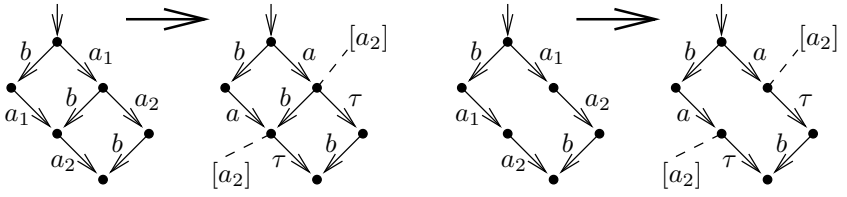
$$\begin{aligned} \rightarrow \subseteq & \left\{ ((s, R), \alpha, (s', R \oplus [v])) \,\middle|\, s \stackrel{\gamma}{\rightarrow} s', \alpha \in \mathrm{adom}(r), r(\alpha) \stackrel{\gamma}{\rightarrow} v \right\} \\ & \cup \left\{ ((s, R), \tau, (s', R')) \,\middle|\, s \stackrel{\gamma}{\rightarrow} s', R \stackrel{\gamma}{\rightarrow} R' \right\} \\ & \cup \left\{ ((s, R), \gamma, (s', R)) \,\middle|\, s \stackrel{\gamma}{\rightarrow} s', \gamma \notin \mathrm{arng}(r) \right\}. \end{aligned}$$

Moreover, the following saturation conditions are required to hold for all $(s, R) \in S$:

1. if $(s, R) \xrightarrow{\alpha} (s', R')$ for $\alpha \in \mathrm{adom}(r)$, $R = \emptyset$ and $r(\alpha) \stackrel{\sigma\surd}{\Longrightarrow}$, then $s \stackrel{\sigma}{\Rightarrow} s''$ such that $(s, R) \stackrel{\alpha}{\Rightarrow} (s'', \emptyset) \approx (s', R')$;
2. if $s \xrightarrow{\gamma} s'$ then either $\exists \alpha \in \mathrm{adom}(r)\colon r(\alpha) \xrightarrow{\gamma} v$, $(s, R) \xrightarrow{\alpha} (s', R \oplus [v])$, or $\exists R \xrightarrow{\gamma} R'\colon (s, R) \xrightarrow{\tau} (s', R')$, or $\gamma \notin \mathrm{arng}(r)$ and $(s, R) \xrightarrow{\gamma} (s', R)$;
3. if $R \xrightarrow{\gamma} R'$ then $\exists s \stackrel{\gamma}{\Rightarrow} s'$ such that $(s, R) \stackrel{\varepsilon}{\Rightarrow} (s', R') \approx (s, R)$.

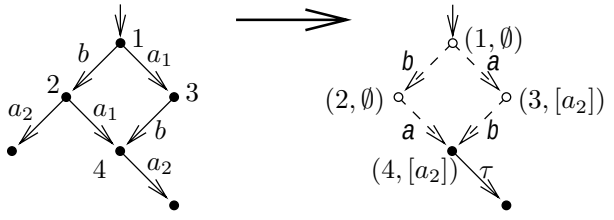
Before showing the formal connection between abstraction and vertical bisimulation, we give a few examples.

EXAMPLE 5.1. Let $r\colon a \mapsto a_1; a_2$. Two easy examples of abstraction are given by the following transition systems (where we only show the nonempty residual sets).

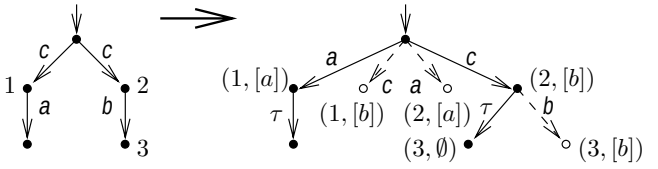


Roughly speaking, the algorithm to follow in order to produce the abstraction of an implementation U, up to some refinement function r, is as follows: first, build all the states and transitions that are reachable from the initial state $(q_U, \emptyset)$; then check if the saturation conditions are satisfied. If a state does not satisfy one of these three conditions, then remove it, together with the transitions that reach it and depart from it. If also $(q_U, \emptyset)$ is removed in this way, then there is no abstraction of U. (Below we give a more precise definition of the algorithm for the special case where r is distinct.)

EXAMPLE 5.2. The following transition system has no abstraction up to $r\colon a \mapsto a_1; a_2$. Consider: if T is an r-abstraction, then the abstract state $(2, \emptyset)$ violates condition 2 and $(3, [a_2])$ violates condition 3 above, and hence neither is in S_T; therefore, $(1, \emptyset)$ does not satisfy condition 1. Hence the initial state is not in S_T, contradicting the assumptions. ("Invalid" states are depicted by open circles, and invalid transitions by dashed arrows.)

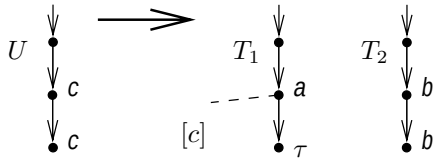


EXAMPLE 5.3. Abstraction becomes more complex if the "explanation" of a transition is ambiguous: finding the correct explanation involves generating all potential ones at first, and then cutting away all those that violate any of the saturation conditions of the definition. For instance, if $r\colon a \mapsto c; a,\ b \mapsto b,\ c \mapsto c; b$ then the following abstraction holds:



Due to ambiguity, it may also be the case that there are several abstractions of a given transition system;

for instance, both T_1 and T_2 are abstractions of U up to $r: a \mapsto c; c, b \mapsto c$:

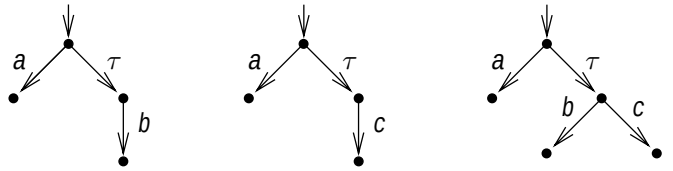


The following theorem states that vertical bisimulation is implied by rooted weak bisimulation w.r.t. some abstraction.

THEOREM 5.1. *$T \lesssim^r U$ if there exists an r-abstraction V of U such that $T \simeq V$.*

The proof can be found in [38]. Although the principle of abstraction strengthens the intuition behind vertical bisimulation, it does not yet offer an easier method of checking vertical bisimulation. After all, as Example 5.2 shows, the abstraction of a transition system is not always defined, and as Example 5.3 shows, it may not be unique when it is defined, and may be nontrivial to construct even when unique. The construction consists of first guessing a solution, i.e., an "explanation" of the transitions of the low-level system, and only afterward checking its correctness w.r.t. the saturation conditions. The latter is hard to do "on the fly" because of saturation conditions 1 and 3, which require weak bisimilarity between certain states of the abstraction. Even worse, abstraction is not a necessary condition for vertical bisimulation, as the following example shows.

EXAMPLE 5.4. Let $r: b \mapsto d,\ c \mapsto d$, and let $t = a + b + \tau; c$, $u_1 = a + d + \tau; d$, and $u_2 = a + \tau; d$. It is easily seen that $t \lesssim^r u_1 \simeq u_2$, and hence also $t \lesssim^r u_2$ due to Rule R_3. However, u_2 has no r-abstraction that is observationally congruent to t; indeed, the possible r-abstractions of u_2 are given by the following transition systems, none of which are observationally equivalent to t. (The pending refinements are empty everywhere.)



On the other hand, it is not difficult to see that for specific classes of refinement functions the problem becomes much easier. In particular, this is true for the class of *distinct* refinement functions introduced in Definition 2.3. The reason for this follows from the next lemma.

LEMMA 5.1. *Let $r: \mathbf{A} \to \mathbf{R_C}$ be a distinct refinement function; let $R \in rsd(r)$.*

1. *If $r(\alpha) \xrightarrow{\gamma} v$ and $r(\alpha') \xrightarrow{\gamma} v'$, then $\alpha = \alpha'$ and $v = v'$.*
2. *If $R \xrightarrow{\gamma} R'$ and $R \xrightarrow{\gamma} R''$ then $R' = R''$.*
3. *If $r(\alpha) \xrightarrow{\gamma}$ then $R \not\xrightarrow{\gamma}$.*

It follows that for any transition $s_U \xrightarrow{\gamma} s'_U$ and any residual set $R \in rsd(r)$, if r is distinct then there is at most one abstracted transition $(s_U, R) \xrightarrow{\alpha} (s'_U, R')$ satisfying the construction in Definition 5.1. Hence, if T is an abstraction of U up to a distinct r, then the transition relation $\to$ of T actually *equals* the set constructed in Definition 5.1, instead of being a subset.

Distinctness of the refinement function ensures that the existence of an abstraction is a necessary condition for the existence of a vertical specification; in fact, the two are bound to be rooted bisimilar. In other words, for distinct refinement functions the inverse direction of Theorem 5.1 also holds. The proof can be found in [38].

THEOREM 5.2. *If $T \lesssim^r U$ for a distinct r, then there exists an r-abstraction V of U, and $T \simeq V$.*

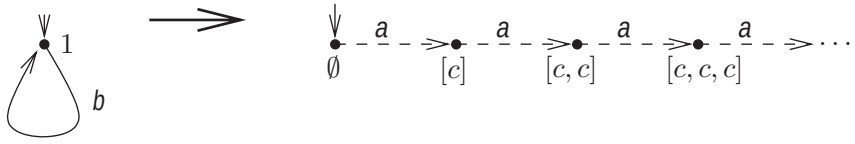
To abstract a given transition system with respect to a distinct refinement function, the only remaining problem is to check whether the saturation conditions of Definition 5.1 can be fulfilled, i.e., if an appropriate relation between pending refinement lists and states exists. Fortunately, this, too, is much

easier than for the general case. First, note that $r(a)$ gives rise to a finite-state transition system for all actions $a \in \mathbf{A}$. The next observation (the proof of which can be found in [38]) is that the abstraction of a finite-state system up to a distinct refinement function is bound to be finite-state.

LEMMA 5.2. *If r is distinct and U is finite-state with r-abstraction T, then T is finite-state.*

Now we turn to the matter of actually constructing an abstraction up to a distinct refinement function. Although this is not a really difficult task, one does have to be a bit careful: even if the abstraction itself (if one exists) is known to be finite, the *potential* abstractions are not, as the following example shows.

EXAMPLE 5.5. The following transition system has no abstraction up to $r: a \mapsto b; c$, but an infinite potential abstraction, if we take each subsequent b-transition to "open" another a-refinement. The resulting states $(1, n * [c])$ violate Condition 3 of Definition 5.1, but if we would try to generate all potential abstractions before checking their saturation properties, the algorithm may not terminate.



It follows that if we are not careful about the order in which to create states of the abstraction and check the saturation conditions, it may be that the algorithm does not terminate in case an abstraction does not exist. The following algorithm builds the abstraction *incrementally* on the size of the residual set, checking the intermediate result before continuing with the next step.

DEFINITION 5.2. Let U be a transition system with $L_U = \mathbf{C}_{\tau,\surd}$, and let $r: \mathbf{A} \to \mathbf{R}_\mathbf{C}$ a distinct refinement function. The *pre*^{k}-abstraction of U, with $k \in \mathbb{N} \cup \{\omega\}$, is the transition system $\langle \mathbf{A}_{\tau,\surd}, S, \to, (q_U, \emptyset)\rangle$, where $S \subseteq S_U \times rsd(r)$ and $\to \subseteq S \times \mathbf{A}_{\tau,\surd} \times S$ are the smallest sets satisfying

- $(q_U, \emptyset) \in S$;
- for all $(s, R) \in S$ and $s \xrightarrow{\gamma} s'$:
  —if $|R| < k$ and $r(\alpha) \xrightarrow{\gamma} v$ with $\alpha \in \text{adom}(r)$, then $(s, R) \xrightarrow{\alpha} (s', R \oplus [v])$ $(\in S)$;
  —if $R \xrightarrow{\gamma} R'$, then $(s, R) \xrightarrow{\tau} (s', R')$ $(\in S)$;
  —if $\gamma \notin \text{arng}(r)$, then $(s, R) \xrightarrow{\gamma} (s', R)$ $(\in S)$.
- if $(s, R) \in S$ and $R \xrightarrow{\gamma} R'$, then $(s, R) \xrightarrow{\gamma} (s, R')$ $(\in S)$.

A pre^k-abstraction is called

- *saturated* if for all $(s, R) \in S$ and $s \xrightarrow{\gamma} s'$, either $\exists \alpha \in \text{adom}(r)$: $r(\alpha) \xrightarrow{\gamma}$ or $R \xrightarrow{\gamma}$ or $\gamma \notin \text{arng}(r)$.
- *consistent* if the following conditions hold:
  —if $(s, \emptyset) \xrightarrow{\alpha} (s', [v])$ and $r(\alpha) \stackrel{\sigma\surd}{\Longrightarrow}$, then $s \stackrel{\sigma}{\Rightarrow} s''$ such that $(s', [v]) \approx (s'', \emptyset)$;
  —if $(s, R) \in S$ and $R \xrightarrow{\gamma} R'$, then $s \stackrel{\gamma}{\Rightarrow} s'$ such that $(s, R) \approx (s', R')$.

The pre$^\omega$-abstraction of U is also simply called its preabstraction.

Note that the preabstraction of a finite-state transition system need not be finite-state; see Example 5.5. The following observations are easy to check. Assume that r is distinct, and let the pre^k-abstractions of U $(k \in \mathbb{N} \cup \{\omega\})$ be given by U^k.

- If U^ω is saturated and consistent, then it is an r-abstraction of U;
- If U has an r-abstraction, then U^ω is saturated and consistent;
- If U^ω is saturated, then all U^k $(k \in \mathbb{N})$ are saturated;
- If $U^k = U^{k+1}$, then $U^k = U^\omega$;
- If U is finite-state and U^ω is saturated, then $\exists n \in \mathbb{N}$: $\forall k \geq n$: $U^\omega = U^k$.

The latter observation is shown using an analogous proof to that of Lemma 5.2. Example 5.5 shows that it does *not* hold if U^ω is not saturated. This brings us to the following algorithm to construct an r-abstraction of U; we will denote the constructed system $U \Uparrow r$.

ALGORITHM 1.

Let $r: \mathbf{A} \to \mathbf{R_C}$ be a distinct refinement function and U a finite-state transition system with $L_U = \mathbf{C}_{\tau,\checkmark}$.

1. Initially, let $k := 0$.
2. Construct the pre^k-abstraction U^k.
3. If U^k is not saturated, the algorithm fails.
4. If $k = 0$ or $U^k \neq U^{k-1}$, let $k := k + 1$ and go back to Step 2.
5. The algorithm succeeds iff U^k is consistent, with outcome $U \Uparrow r = U^k$.

The correctness of the algorithm is formulated in the following theorem, which follows from the observations above.

THEOREM 5.3. *Let $r: \mathbf{A} \to \mathbf{R_C}$ be a distinct refinement function and U a finite-state transition system with $\Lambda_U = \mathbf{C}_{\tau,\checkmark}$. Algorithm* 1 *succeeds iff T has an r-abstraction, which is then given by $U \Uparrow r$.*

6. OPEN TERMS

So far, we have restricted the proof system for vertical implementation to closed terms only. As a consequence, there is no proof-theoretic way to deduce vertical implementation of recursive terms. Since this is a severe restriction to the usefulness of the theory, in this section we consider its extension to open terms.

6.1. *Implementation Environments*

The prime candidate proof rule for a (horizontal) implementation relation over recursive terms is the *recursion congruence* property

$$\frac{t \sqsubseteq u}{\mu x.\, t \sqsubseteq \mu x.\, u}.$$

The premise of this rule is a statement concerning open terms. Since an implementation relation $\leq$ is usually only defined directly over closed terms, to apply the rule one must extend the relation. The standard open term extension is defined by

$$t \leq u :\Leftrightarrow \forall f: \mathrm{fv}(t, u) \to \mathbf{L}: t\langle f \rangle \leq u\langle f \rangle. \tag{2}$$

(See [36] for alternative ways of extending relations to open terms.) As for the proof system, rather than turning the above definition into a proof rule (which would not be finitary), one can at least provide the following reflexivity axiom for open terms:

$$\overline{x \sqsubseteq x}.$$

Now let us consider the appropriate generalization to vertical implementation. In order to interpret $t \leq^r u$ where t and u are open terms, we must take into account that x in t lives in the abstract world, whereas x in u lives in the concrete world. This means that we must allow *different* terms to be substituted for x on the left- and right-hand sides; in fact, the term substituted on the right-hand side should itself be an implementation of the term substituted on the left-hand side. In other words, the relation $x \leq^r x$ should not be interpreted to mean $t \leq^r t$ for arbitrary t (which certainly does not hold in general) but rather $t \leq^r u$ for arbitrary t, u such that $t \leq^r u$ (which is trivially true). However, but this interpretation still poses problems, as the following example shows.

EXAMPLE 6.1. Consider the refinement function $r: a \mapsto a_1; a_2$. With the rules above we can derive

$$\dfrac{\dfrac{x \sqsubseteq^r x}{x/a \sqsubseteq^{id} x/a_1, a_2}\mathrm{R}_{10}}{x/a \sqsubseteq x/a_1, a_2}\mathrm{R}_2.$$

This conclusion is not correct for any reasonable implementation relation $\leq$: after instantiation of x, the terms x/a and $x/a_1, a_2$ are in general not even related up to trace inclusion.

Here, the error lies in the fact that the relation $x \leq^r x$ is interpreted as meaning that $x\langle t/x\rangle \leq^r x\langle u/x\rangle$ (i.e., $t \leq^r u$) holds for all t, u such that $t \leq^r u$ (which is trivially true), whereas $x/a \leq^{id} x/a_1, a_2$ is taken to mean that $(x/a)\langle t/x\rangle \leq^{id} (x/a_1, a_2)\langle u/x\rangle$ (i.e., $t/a \leq^{id} u/a_1, a_2$) holds for all t, u *such that* $t \leq^{id} u$; i.e., the substitutions considered for x have implicitly changed. In other words, the reason for the error is that the derivation rule R_{10} changes the refinement function, whereas the assumption about the variable should *not* be changed.

This problem can be solved by making the assumptions about free variables explicit. For this purpose, we introduce *implementation environments* Γ, which are lists $x_1 : r_1, \ldots, x_n : r_n$ (where $x_i = x_j$ implies that $i = j$). We denote $\mathrm{dom}\,\Gamma = \{x_1, \ldots, x_n\}$ and $\Gamma(x_i) = r_i$ for all $1 \leq i \leq n$. Each pair $(x_i : r_i) \in \Gamma$ expresses the assumption that x_i in the concrete system implements x_i in the abstract system up to the refinement function r_i. Correspondingly, a vertical implementation relation over open terms consists not of statements of the form $t \leq^r u$, but rather of statements of the form $t \leq^r_\Gamma u$, with $\mathrm{fv}(t, u) \subseteq \mathrm{dom}\,\Gamma$. For instance, $t \leq^{r'}_{x:r} u$ expresses that *if* x on the right-hand implements x on the left-hand side up to r, *then* u implements t up to r'.

For the proof-theoretic counterpart to the actual relation $t \leq^r_\Gamma u$, we use the notation $\Gamma \vdash t \sqsubseteq^r u$. For instance, the correct version of the judgement derived in Example 6.1 is $x:r \vdash x/a \sqsubseteq^{id} x/a_1, a_2$. If $\mathrm{dom}\,\Gamma = \emptyset$, meaning that t and u are closed terms, we write $\vdash t \sqsubseteq^r u$ or simply $t \sqsubseteq^r u$ as before. We abbreviate multiple statements $\Gamma \vdash t_i \sqsubseteq^{r_i} u_i$ for $i = 1, 2$ to $\Gamma \vdash t_1 \sqsubseteq^{r_1} u_1, t_2 \sqsubseteq^{r_2} u_2$.

Using implementation environments, we can now formulate the proof rules for open terms, including recursion congruence. The existing proof rules of vertical bisimulation over **L**, presented in Table 3, must be extended with environments as well. The result is given in Table 5. The new rules are R_{25} and R_{26}. It is straightforward to show that the following properties hold:

PROPOSITION 6.1.

1. *If* $\Gamma \vdash t \sqsubseteq^r u$ *and* $\mathrm{dom}\,\Gamma \cap \mathrm{dom}\,\Gamma' = \emptyset$, *then* $\Gamma, \Gamma' \vdash t \sqsubseteq^r u$ (weakening).
2. *If* $\Gamma, x:r' \vdash t \sqsubseteq^r u$ *and* $\Gamma \vdash t' \sqsubseteq^{r'} u'$, *then* $\Gamma \vdash t\langle t'/x\rangle \sqsubseteq^r u\langle u'/x\rangle$ (substitutivity).

TABLE 5

Vertical Proof Rules for Open and Recursive Terms

$$\dfrac{}{\mathrm{fv}(t):id \vdash t \sqsubseteq^{id} t}\mathrm{R}_{13} \qquad \dfrac{X:id \vdash t \sqsubseteq^{id} u}{t \sqsubseteq u}\mathrm{R}_{14} \qquad \dfrac{t \sqsubseteq t' \quad \Gamma \vdash t' \sqsubseteq^r u' \quad u' \sqsubseteq u}{\Gamma \vdash t \sqsubseteq^r u}\mathrm{R}_{15}$$

$$\dfrac{}{\Gamma \vdash \mathbf{0} \sqsubseteq^r \mathbf{0}}\mathrm{R}_{16} \qquad \dfrac{}{\Gamma \vdash \mathbf{1} \sqsubseteq^r \mathbf{1}}\mathrm{R}_{17} \qquad \dfrac{}{\Gamma \vdash \alpha \sqsubseteq^r r(\alpha)}\mathrm{R}_{18} \qquad \dfrac{\Gamma \vdash t_1 \sqsubseteq^r u_1,\ t_2 \sqsubseteq^r u_2}{\Gamma \vdash t_1 + t_2 \sqsubseteq^r u_1 + u_2}\mathrm{R}_{19}$$

$$\dfrac{\Gamma \vdash t_1 \sqsubseteq^r u_1,\ t_2 \sqsubseteq^r u_2}{\Gamma \vdash t_1; t_2 \sqsubseteq^r u_1; u_2}\mathrm{R}_{20} \qquad \dfrac{\Gamma \vdash t \sqsubseteq^r u \quad \phi \upharpoonright \mathrm{adom}(r) = id_{\mathrm{adom}(r)}}{\Gamma \vdash t[\phi] \sqsubseteq^r u[\phi]}\mathrm{R}_{21}$$

$$\dfrac{\Gamma \vdash t \sqsubseteq^r u \quad r \text{ preserves } A}{\Gamma \vdash t/A \sqsubseteq^{r\setminus A} u/\mathcal{A}(r(A))}\mathrm{R}_{22} \qquad \dfrac{\Gamma \vdash t_1 \sqsubseteq^r u_1,\ t_2 \sqsubseteq^r u_2 \quad r \text{ is distinct on } A}{\Gamma \vdash t_1 \parallel_A t_2 \sqsubseteq^r u_1 \parallel_{\mathcal{A}(r(A))} u_2}\mathrm{R}_{23}$$

$$\dfrac{r(a) = u_1; u_2 \quad \Gamma \vdash t \sqsubseteq^r v}{\Gamma \vdash a; t \sqsubseteq^r u_1; (u_2 \parallel\!\mid v)}\mathrm{R}_{24}$$

$$\dfrac{x \notin \mathrm{dom}\,\Gamma}{\Gamma, x:r \vdash x \sqsubseteq^r x}\mathrm{R}_{25} \qquad \dfrac{\Gamma, x:r \vdash t \sqsubseteq^r u}{\Gamma \vdash \mu x.\, t \sqsubseteq^r \mu x.\, u}\mathrm{R}_{26}$$

EXAMPLE 6.2. Let $r: a \mapsto a_1; a_2$. The following is a derivation of $\mu x.\, a; x \mid\mid\mid b \sqsubseteq^r \mu x.\, a_1; a_2; x \mid\mid\mid b$:

$$\dfrac{\dfrac{\dfrac{}{x{:}r \vdash a \sqsubseteq^r a_1; a_2}\mathrm{R}_{18} \quad \dfrac{}{x{:}r \vdash x \sqsubseteq^r x}\mathrm{R}_{25}}{x{:}r \vdash a; x \sqsubseteq^r a_1; a_2; x}\mathrm{R}_{20} \quad \dfrac{}{x{:}r \vdash b \sqsubseteq^r b}\mathrm{R}_{18}}{x{:}r \vdash a; x \mid\mid\mid b \sqsubseteq^r a_1; a_2; x \mid\mid\mid b}\mathrm{R}_{23}}{\vdash \mu x.\, a; x \mid\mid\mid b \sqsubseteq^r \mu x.\, a_1; a_2; x \mid\mid\mid b}\mathrm{R}_{26}.$$

6.2. *Vertical Bisimilarity of Open Terms*

Above, we defined the standard open term extension of a given closed-term relation $\leq$ (see (2)). There is a corresponding extension of closed-term vertical bisimilarity $\lesssim^r$ to open-term vertical bisimilarity $\lesssim^r_\Gamma$, based on investigating arbitrary Γ-respecting closed instantiations of the free variables. The definition is

$$t \lesssim^r_\Gamma u :\Leftrightarrow (\forall f, g{:}\, \mathrm{fv}(t, u) \rightharpoonup \mathbf{L}{:}\ f \lesssim^\Gamma g \Rightarrow t\langle f \rangle \lesssim^r u\langle g \rangle), \tag{3}$$

where $f \lesssim^\Gamma g$ abbreviates $\forall x \in \mathrm{dom}(f) = \mathrm{dom}(g) = \mathrm{dom}\,\Gamma{:}\ f(x) \lesssim^{\Gamma(x)} g(x)$. Comparing (3) with (2), a prominent difference is the fact that in the extension of vertical bisimilarity, the terms on the left- ("abstract") and the right-hand ("concrete") sides are instantiated with different (albeit related) functions f and g. This is necessitated by the fact that both sides of the relation live on different levels of abstraction. For instance, Rule R_{25} in Table 5 could not be satisfied if we would use the same substitutions for the abstract and concrete instances of the variable x.

Unfortunately, the definition in (3) has as a consequence that the congruence of vertical bisimilarity with respect to recursion, as formulated in Rule R_{26} of Table 5, is quite difficult to prove. The usual proof techniques for this kind of congruence property, such as the *up-to* technique proposed in [26, 30] or Howe's technique used in functional calculi [26, 40], are not applicable to nonreflexive, nontransitive relations like vertical bisimilarity. Furthermore, we have investigated alternative extensions of closed relations to open terms in [36]; however, the resulting theory is neither developed far enough nor simple enough to apply here. For that reason, we resort to the sublanguage of *strictly well-guarded terms*; with this sublanguage, which gives rise to image-finite systems only (w.r.t. the weak transition relation), we can use an inductive characterization of vertical bisimulation that makes it possible to prove the desired recursion congruence property.

DEFINITION 6.1. First we define when an arbitrary term (in $\mathbb{L}$) is called *a strict guard*:

- $\mathbf{0}$ and a are strict guards (for all $a \in \mathbf{A}$);
- $t + u$ is a strict guard if both t and u are strict guards;
- $t; u$ and $t \|_A u$ are strict guards if either t or u is a strict guard;
- $t[\phi]$ and $\mu x.\, t$ are strict guards if t is a strict guard;
- $\mathbf{1}$, τ, t/A and x are *not* strict guards.

Next we define when a variable is called *strictly guarded* in an arbitrary term:

- x is strictly guarded in $\mathbf{0}$, $\mathbf{1}$ and α (for all $\alpha \in \mathbf{A}_\tau$);
- x is strictly guarded in $t + u$ and $t \|_A u$ if x is strictly guarded in both t and u;
- x is strictly guarded in $t; u$ is guarded if x is strictly guarded in t and either t is a strict guard or x is strictly guarded in u;
- x is strictly guarded in $t[\phi]$ and $\mu x.\, t$ if x is strictly guarded in t;
- x is *not* strictly guarded in t/A;
- x is strictly guarded in y ($\in \mathbf{X}$) if $x \neq y$.

We call $t \in \mathbb{L}$ *strictly well guarded* if for all subterms $\mu x.\, u$ of t, x is strictly guarded in u and $\mu x.\, u$ occurs outside the context of any hiding operator.

The set of strictly well guarded terms will be denoted $\mathbf{L}_{\mathbf{A}}^{\mathsf{swg}}$ (and accordingly $\mathbf{L}_{\mathbf{A}}^{\mathsf{swg}}$ for the closed fragment); however, where this does not give rise to confusion we will drop the superscript $^{\mathsf{swg}}$ and implicitly assume all terms to be strictly well guarded.

Clearly, if x is strictly guarded in t then x is guarded in t (see Definition 2.1), and if t is strictly well guarded then t is well guarded. Strict guardedness satisfies the same preservation properties under substitution as guardedness; the following proposition is the counterpart of Proposition 2.1.

PROPOSITION 6.2. *Let $t, u \in \mathbb{L}$ and $x, y \in \mathbf{X}$.*

1. *If x is strictly guarded in t and u, then x is strictly guarded in $t\langle u/y\rangle$.*
2. *If t and u are strictly well guarded, then so is $t\langle u/y\rangle$.*

The following proposition states the operational consequences: strict guardedness is preserved by internal transitions, and strict well-guardedness is preserved by any transition (just like ordinary well-guardedness; compare Proposition 2.2).

PROPOSITION 6.3. *Let $t \in \mathbb{L}$.*

1. *If x is strictly guarded in t and $t \xrightarrow{\tau} t'$, then x is strictly guarded in t'.*
2. *If t is strictly well guarded and $t \xrightarrow{\alpha} t'$, then t' is strictly well guarded.*

A transition system is called *strict image-finite* if for all $s \in S$ and $a \in A$, the number of states s' such that $s \stackrel{a}{\Rightarrow} s'$ is finite. The following property formalizes the claim, already made above, that strict guardedness is enough to guarantee strict image-finiteness.

PROPOSITION 6.4. *For all $\mathbf{A} \subseteq \mathbf{U}$, $\langle \mathbf{A}_{\checkmark,\tau}, \mathbf{L}_{\mathbf{A}}^{\mathsf{swg}}, \rightarrow\rangle$ is a strict image-finite LTS.*

Stratified Vertical Bisimilarity. The soundness proof of the congruence rule for recursion is based on an inductive characterization of vertical bisimilarity, applying the principle of stratification seen in [29], except that—due to the strict image-finiteness of the systems we consider—we need only countably many approximations.

THEOREM 6.1. *If T is a strict image-finite transition system, then $\preccurlyeq^r = \bigcap_{i\in\mathbb{N}} \preccurlyeq_i^r$ and $\lesssim^r = \bigcap_{i\in\mathbb{N}} \lesssim_i^r$, where $(\preccurlyeq_i^r)_{i\in\mathbb{N}}$ and $(\lesssim_i^r)_{i\in\mathbb{N}}$ are stratifications of weak and rooted vertical bisimilarity defined as follows*:

- $\preccurlyeq_0^r = S \times S \times rsd(r)$ *and* $\lesssim_0^r = S \times S$;
- *For all $i > 0$, $\preccurlyeq_i^r \subseteq S \times S \times rsd(r)$ is the largest ternary relation such that for all $s_1 \preccurlyeq_i^{r,R} s_2$:*
  - *—if $R = \emptyset$ and $s_1 \stackrel{\alpha}{\Rightarrow} s_1'$, then one of the following holds*:
    1. *$\alpha \in \mathrm{adom}(r)$, and $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$ with $|\sigma| < i$ implies that $\exists s_2 \stackrel{\sigma}{\Rightarrow} s_2'$ such that $s_1' \preccurlyeq_{i-|\sigma'|}^{r,\emptyset} s_2'$.*
    2. *$\alpha \notin \mathrm{adom}(r)$ and $\exists s_2 \stackrel{\hat{\alpha}}{\Rightarrow} s_2'$ such that $s_1' \preccurlyeq_{i-1}^{r,\emptyset} s_2'$.*
  - *—if $s_2 \stackrel{\gamma}{\Rightarrow} s_2'$ then one of the following holds*:
    1. *$\exists \alpha \in \mathrm{adom}(r).\ \exists s_1 \stackrel{\alpha}{\Rightarrow} s_1'$ and $\exists r(\alpha) \xrightarrow{\gamma} v$ such that $s_1' \preccurlyeq_{i-1}^{r,R\oplus[v]} s_2'$.*
    2. *$\exists s_1 \stackrel{\varepsilon}{\Rightarrow} s_1'$ and $\exists R \xrightarrow{\gamma} R'$ such that $s_1' \preccurlyeq_{i-1}^{r,R'} s_2'$;*
    3. *$\gamma \notin \mathrm{arng}(r)$ and $\exists s_1 \stackrel{\hat{\gamma}}{\Rightarrow} s_1'$ such that $s_1' \preccurlyeq_{i-1}^{r,R} s_2'$.*
  - *—If $R \xrightarrow{\gamma} R'$ then $\exists s_2 \stackrel{\gamma}{\Rightarrow} s_2'$ such that $s_1 \preccurlyeq_{i-1}^{r,R'} s_2'$.*
- *For all $i > 0$, $\lesssim_i^r \subseteq S \times S$ is the largest biroot of $\preccurlyeq_{i-1}^{r,\emptyset}$.*

The proof is not essentially different from the case for standard (rooted) weak bisimilarity; it can be found in [38]. We now show that $t \lesssim_{x:r}^r u$ implies that $\mu x.\, t \lesssim_i^{r,\emptyset} \mu x.\, u$ for all $i \in \mathbb{N}$; this then implies that $\mu x.\, t \lesssim^r \mu x.\, u$. For this purpose, we define the *approximations* of a recursive term $\mu x.\, t$, in the standard way:

$$\mu^0 x.\, t = \mathbf{0}$$
$$\mu^{i+1} x.\, t = t\langle \mu^i x.\, t/x\rangle.$$

By induction on i and the fact that $\mathbf{0} \lesssim^r \mathbf{0}$, by the definition of $\lesssim^r$ over open terms (see (3)) it follows

that $t \lesssim^r_{x:r} u$ implies that $\mu^i x.t \lesssim^r \mu^i x.u$ for all $i \in \mathbb{N}$. The precise relation between a recursive term and its approximants can be captured by yet another stratification, this time of strong bisimilarity.

DEFINITION 6.2. Let T be a transition system. For all $i \in \mathbb{N}$, the relation $\sim_i \subseteq S \times S$ is defined as follows:

- $\sim_0 = S \times S$;
- if $i > 0$, then $\sim_i \subseteq S \times S$ is the largest relation such that for all $s_1 \sim_i s_2$:
  —if $s_1 \xrightarrow{\alpha} s_1'$, then $s_2 \xrightarrow{\alpha} s_2'$ such that $s_1' \sim_{i-1} s_2'$, and $s_1' \sim_i s_2'$ if $\alpha = \tau$;
  —if $s_2 \xrightarrow{\alpha} s_2'$, then $s_1 \xrightarrow{\alpha} s_1'$ such that $s_1' \sim_{i-1} s_2'$, and $s_1' \sim_i s_2'$ if $\alpha = \tau$.

Note that the above definition is unusual in that the stratification depth is *not* decreased for internal actions. This again has to do with the strict image-finiteness of the systems we consider. We now need two auxiliary lemmas (the proofs of which can be found in the Appendix). The first one states that a sufficient condition for stratified vertical bisimilarity is to compose stratified strong bisimilarity, then ordinary vertical bisimilarity, and then stratified strong bisimilarity again.

LEMMA 6.1. *For all $i > 0$, the following inequalities hold:*

1. $\sim_i \circ \preccurlyeq^{r,R} \circ \sim_i \subseteq \preccurlyeq^{r,R}_{i-1}$ *for all* $R \in rsd(r)$;
2. $\sim_i \circ \lesssim^r \circ \sim_i \subseteq \lesssim^r_{i-1}$.

The following lemma states that every approximation of a recursive term is related to the actual recursive term up to a stratification depth equal to the approximation depth.

LEMMA 6.2. *Let $t \in \mathbb{L}^{\mathsf{swg}}$ with* $\mathrm{fv}(t) \subseteq \{x\}$. *For all $i \in \mathbb{N}$, $\mu x.t \sim_i \mu^i x.t$.*

We are now ready to prove the desired recursion congruence property of rooted vertical bisimilarity.

THEOREM 6.2. *Let $t, u \in \mathbb{L}^{\mathsf{swg}}$. If $t \lesssim^r_{\Gamma,x:r} u$, then $\mu x.t \lesssim^r_\Gamma \mu x.u$.*

Proof. First we treat the case where $\Gamma = \emptyset$, i.e., $\mathrm{fv}(t,u) \subseteq \{x\}$. For arbitrary i, using Lemma 6.2 we have

$$\mu x.t \sim_i \mu^i x.t \lesssim^r \mu^i x.u \sim_i \mu x.u.$$

By Lemma 6.1.2, it follows that $\mu x.t \lesssim^r_i \mu x.u$ for all $i \in \mathbb{N}$. According to Theorem 6.1, therefore, $\mu x.t \lesssim^r \mu x.u$. For arbitrary Γ, the theorem follows from the fact that if f maps to closed terms and $x \notin \mathrm{dom}(f)$, then $(\mu x.t)\langle f\rangle = \mu x.t\langle f\rangle$ and $(\mu^i x.t)\langle f\rangle = \mu^i x.t\langle f\rangle$ for all $i \in \mathbb{N}$. ∎

We can now state the main result of this paper, namely the soundness of the derivation rules for open terms in Table 5 with respect to vertical bisimilarity. The proof is given in the Appendix.

THEOREM 6.3. *Rooted vertical bisimilarity satisfies all the rules in Table* 5.

It is also not difficult to see that the following semantic counterpart to Proposition 6.1 holds:

PROPOSITION 6.5.

1. *If $t \lesssim^r_\Gamma u$ and* $\mathrm{dom}\,\Gamma \cap \mathrm{dom}\,\Gamma' = \emptyset$, *then $t \lesssim^r_{\Gamma\cup\Gamma'} u$.*
2. *If $t \lesssim^r_\Gamma u$ and $f(x) \lesssim^{\Gamma(x)}_{\Gamma'} g(x)$ for all $x \in \mathrm{dom}\,\Gamma$, then $t\langle f\rangle \lesssim^r_{\Gamma'} u\langle g\rangle$.*

This is proved by applying the definition of the open term extension $\lesssim^r_\Gamma$, given in (3).

As a final result, note that Corollary 4.1, concerning the correctness of syntactic refinement modulo vertical bisimilarity, can immediately be generalized to the language with recursion. See Table 4 for the defination of $r^*: \mathbb{L} \rightharpoonup \mathbb{L}$.

COROLLARY 6.1. *For all $t \in \mathbb{L}_{\mathbf{A}}$ and $r: \mathbf{A} \to \mathbf{R}$, if r^* is defined on t then $t \lesssim^r_{\mathrm{fv}(t):r} r^*(t)$.*

7. EXAMPLES

In this section we apply our theory to a number of examples. First we consider a small data base example used by Brinksma *et al.* in [7]. We then extend this example to demonstrate the principle of weakening sequential composition during refinement. Finally, we consider a refinement-driven design step of a booking agent, inspired by Wehrheim [44], as an example of a nonfinite-state implementation that can nevertheless be proved correct using our proof system.

7.1. *A Distributed Data Base*

The first example concerns a distributed data base that can be queried and updated, and an agent responsible for updating the data base; the latter can alternatively decide to do some local actions not concerning the data base. An important simplification is that the *state* of the data base is completely abstracted away from. Data base and agent are modeled by the transition systems $Data_S$ and $Agent_S$ depicted in Fig. 2.

The problem considered in [7] is to change the interface between data base and agent, so that the two no longer communicate over a single update action; instead, updating consists of two separate stages, in which the update is *requested* and *confirmed*, respectively. In our setting, this can be expressed by a refinement function $r: upd \mapsto req; cnf$. Moreover, it is required that in the meantime (between request and confirmation), querying the data base should not be disabled. The solution proposed is to refine data base and agent by the behavior shown in Fig. 3.

It is seen that, similar to our approach, the implementations proposed in [7] differ from the corresponding specifications in the level of abstraction of their alphabets. The correctness criterion employed in [7] circumvents the associated problems by just requiring (horizontal) correctness *after hiding* the relevant actions: i.e., they prove that

$$(Data_S \parallel_{upd} Agent_S)/upd \leq (Data_I \parallel_{req,cnf} Agent_I)/req, cnf,$$

where $\leq$ is a testing preorder.

The same result holds in our approach (albeit up to rooted bisimilarity); in that sense, we achieve nothing new. However, our method of establishing this result is quite different.

- The first point is that we can state correctness in a more general manner, *before* hiding the actions that are changed; for it is not difficult to see that the following hold:

$$Data_S \lesssim^r Data_I$$
$$Agent_S \lesssim^r Agent_I$$

Moreover, we have an effective way of checking this, through the Abstraction Theorem 5.1, by constructing $Data_I \Uparrow r$ and $Agent_I \Uparrow r$ (see Fig. 4) and observing that $Data_I \Uparrow r \simeq Data_S$ and $Agent_I \Uparrow r \simeq Agent_S$.

- The second point is that we can also prove these vertical inequalities *algebraically*, and in fact *derive* $Data_I$ from $Data_S$ and $Agent_I$ from $Agent_S$. (In the approach of [7], such a derivation is possible for *Data* but not for *Agent*.) Consider the algebraic specifications

$$Data_S = (\mu x.\, qrg; x \,|||\, (\mu y.\, upd; y)$$
$$Agent_S = \mu z.\, upd; z + loc; z$$
$$Data_I = (\mu x.\, qry; x) \,|||\, (\mu y.\, req; cnf; y)$$
$$Agent_I = \mu z.\, req; cnf; z + loc; z.$$

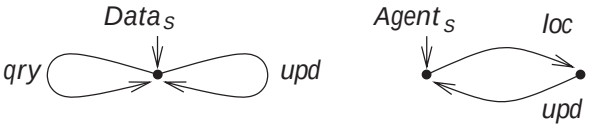


FIG. 2. Specification of data base and agent.

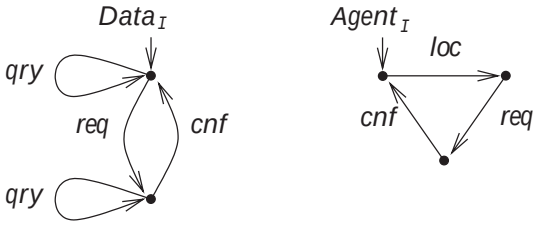


FIG. 3. Implementation of data base and agent.

The correctness of the *Data* part can be shown as

$$\dfrac{\dfrac{\vdots}{\vdash \mu x.\, qry;\, x \sqsubseteq^r \mu x.\, qry;\, x} \qquad \dfrac{\dfrac{\dfrac{}{y\colon r \vdash upd \sqsubseteq^r req;\, cnf}\,\mathrm{R}_{18} \qquad \dfrac{}{y\colon r \vdash y \sqsubseteq^r y}\,\mathrm{R}_{25}}{y\colon r \vdash upd;\, y \sqsubseteq^r req;\, cnf;\, y}\,\mathrm{R}_{20}}{\vdash \mu y.\; upd;\, y \sqsubseteq^r \mu y.\; req;\, cnf;\, y}\,\mathrm{R}_{26}}{\vdash Data_S \sqsubseteq^r Data_I}\,\mathrm{R}_{23}.$$

The correctness of the *Agent* part is proved in an analogous fashion.

- As a final point, the correctness of the combined system again follows by application of algebraic derivation rules:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash Data_S \sqsubseteq^r Data_I} \qquad \dfrac{\vdots}{\vdash Agent_S \sqsubseteq^r Agent_I}}{\vdash Data_S \parallel_{upd} Agent_S \sqsubseteq^r Data_I \parallel_{req,cnf} Agent_I}\,\mathrm{R}_{23}}{\vdash (Data_S \parallel_{upd} Agent_S)/upd \sqsubseteq^{id} (Data_I \parallel_{req,cnf} Agent_I)/req,\, cnf}\,\mathrm{R}_{22}}{(Data_S \parallel_{upd} Agent_S)/upd \sqsubseteq (Data_I \parallel_{req,cnf} Agent_I)/req,\, cnf}\,\mathrm{R}_{14}.$$

Note that we can as easily derive another, incomparable implementation for $Data_S$ by first rewriting its specification to the rooted bisimilar $\mu D.\, qry;\, D + upd;\, D$, and applying syntactic substitution to that term. This results in an equally correct implementation $Data'_I = \mu D.\, qry;\, D + req;\, cnf;\, D$, where the *qry* action is not possible in between *req* and *cnf*.

Refinement-as-Operator. In the "traditional" approach to action refinement, where refinement is treated as an operator, one can also show that $Data_I$ implements $Data_S$ and $Agent_I$ implements $Agent_S$. In fact, the implementations can even be derived algebraically: Reference [21] gives conditions under which syntactic substitution coincides with semantic refinement, and it so happens that these conditions are satisfied in the present example. In the light of this example, the advantages of vertical implementation, already discussed in the Introduction, are the following:

- Our method, being based on interleaving semantics, allows more than one implementation of the abstract transition system, $Data_S$, but not so for traditional action refinement: instead, there a more

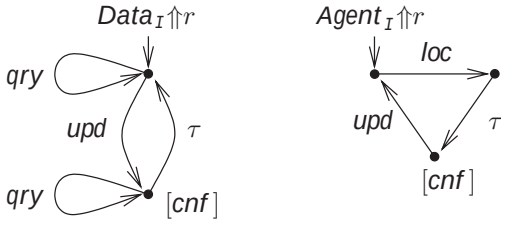


FIG. 4. Abstraction of the data base implementation of Fig. 3.

"precise" specification must be given, either as a term or in a more expressive semantic model. That more precise specification will then allow *either* $Data_I$ *or* $Data'_I$ as an implementation (or possibly yet something different); in no circumstances will it allow both.

- More importantly, our method makes it possible to "collapse" vertical implementation back to horizontal implementation: having derived $Data_I$ and $Agent_I$, we can compose them, hide the interface actions, and get a system that is correct in the well-known, standard interleaving sense with respect to the specification (being the composition of $Data_S$ and $Agent_S$). This means that our notion of vertical implementation can be integrated into existing interleaving-based design methods.

7.2. *The Data Base Revisited: Multiple States*

It is clear that the above is only a toy example; for instance, the data base has only a single state. We now consider a slightly more realistic version in which changes of state are possible. Assume that the state of the data base consists of a natural number in the range $1 - n$, and consider the specification

$$Query_i = \mu x.\, \mathbf{1} + qry_i; x \quad (\text{for } i = 1, \ldots, n)$$
$$Data_S = Query_1; \mu y.\, \left(\sum_{i=1}^{n} upd_i; Query_i \right); y.$$

Hence $Data_S$ specifies that after an update action, where a value is written, any number of consecutive queries can be performed, each of which reads the value just written. Furthermore, for the initial state it is assumed that $i = 1$. For instance, if $n = 2$ then the behavior of $Data_S$ is depicted in Fig. 5.

The refinement consists of splitting the update actions as before:

$$r : upd_i \mapsto req_i; cnf.$$

In the implementation, querying is allowed to overlap with the confirmation phase of the update:

$$Data_I = Query_1; \mu y.\, \left(\sum_{i=1}^{n} req_i; (cnf \;|||\; Query_i) \right); y.$$

The behavior of $Data_I$ for the case $n = 2$ is also shown in Fig. 5. It is straightforward to prove the correctness of $Data_I$ up to r, i.e., $\vdash Data_S \lesssim^r Data_I$. We show the crucial part of the proof:

$$\frac{\overline{y : r \vdash upd_i \sqsubseteq^r req_i; cnf}\,\mathrm{R}_{18} \qquad \begin{array}{c}\vdots\\ \overline{y : r \vdash Query_i \sqsubseteq^r Query_i}\end{array}}{y : r \vdash upd_i \sqsubseteq^r req_i; (cnf \;|||\; Query_i)}\,\mathrm{R}_{24}.$$

Note that, as before, there are many possible implementations of $Data_S$; in particular, the completely

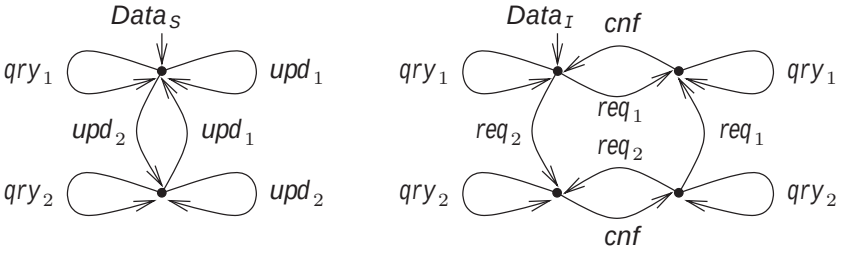


FIG. 5. Specification and implementation of a data base with two states.

sequentialized version is also correct:

$$\vdash Data_S \lesssim^r Query_1; \mu y.\left(\sum_{i=1}^{n} req_i; cnf; Query_i\right); y.$$

7.3. *A Booking Agent*

The final example involves an implementation that is no longer finite-state. We show that our theory can nevertheless be used to prove correctness. Consider a travel agent whose task is to allow customers to book a holiday trip of their choosing. The procedure followed by the agent consists of requesting information about a possible trip, on the basis of which the customer says either yes or no. An abstract description of the agent and a potential customer could look as follows:

$$Agent_S = \mu x.\, info; (no; x + yes; x)$$
$$Cust_S = \mu y.\, info; (\tau; no; (y + \tau) + yes; trip).$$

This particular example customer either agrees to the requested trip immediately or thinks for a bit and decides against it, after which he either tries again or quits. The combined behaviour of agent and customer is described by

$$S = (Agent_S \parallel_A Cust_S)/A,$$

where the synchronization set is given by $A = \{info, yes, no\}$. Note that we have the equivalence

$$S \simeq \tau; (\tau + \tau; trip);$$

that is, the cooperation of customer and travel agent results either in no visible effect or in a trip being taken.

It is a particular, important aspect of booking systems that in between the request for information and the booking decision, no other person may access the same information, since this could result in double bookings. Hence the request for information has an implicit "lock" associated with it. The design step we investigate is to split the "yes" answer into phases: first the trip is actually booked, then the relevant information is either printed directly or mailed to the customer's address. The lock can be released only after booking, but the second phase is independent of it: the next customer can already be helped during the second phase. The "no" action, on the other hand, corresponds to releasing the (lock on the) info. Hence, the implementation is driven by the following refinement function r:

$$yes \mapsto book; (print + mail)$$
$$no \mapsto rel.$$

The proposed implementation is

$$Agent_I = \mu x.\, info; (rel; x + book; ((print + mail) \,|||\, x))$$
$$Cust_I = \mu x.\, info; (\tau; rel; (y + \tau) + book; (print + mail); trip).$$

Note that, since there is no bound on the number of outstanding print or mail actions, $Agent_I$ is infinite-state. Figure 6 sketches its behavior.

Since $Cust_I = r^*(Cust_S)$, it follows by Corollary 6.1 that $Cust_S \lesssim^r Cust_I$. On the other hand, $Agent_I \lesssim^r Agent_S$ can again be shown using the proof system; the crucial part of the proof is

$$\dfrac{\dfrac{\vdots}{x{:}r \vdash no; x \sqsubseteq^r rel; x} \quad \dfrac{\dfrac{}{x{:}r \vdash yes \sqsubseteq^r book; (print + mail)}\,\mathrm{R}_{18} \quad \dfrac{}{x{:}r \vdash x \sqsubseteq^r x}\,\mathrm{R}_{25}}{x{:}r \vdash yes; x \sqsubseteq^r book; ((print + mail) \,|||\, x)}\,\mathrm{R}_{24}}{x{:}r \vdash no; x + yes; x \sqsubseteq^r rel; x + book; ((print + mail) \,|||\, x)}\,\mathrm{R}_{19}.$$

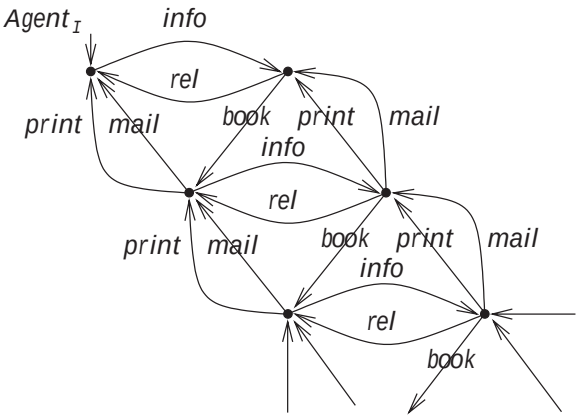


FIG. 6. Implementation of the booking agent.

By R_{23} and R_{22}, it moreover follows that

$$S \simeq I = (Agent_I \|_C Cust_I)/C,$$

where $C = \{info, rel, book, print\}$. Note that we are back to standard rooted bisimilarity here; hence for instance, together with the observation above it follows that

$$I \simeq \tau;(\tau + \tau; trip).$$

This shows once more that vertical implementation, in the sense of this paper, seamlessly fits onto standard (interleaving) correctness criteria.

8. EVALUATION AND FUTURE EXTENSIONS

The method used to relate specifications and implementation belonging to different levels of abstraction proposed in this paper is quite new, and differs from existing theories of action refinement in the following respects:

- We allow a given abstract specification to have different, incomparable implementations under a given, fixed refinement function. This immediately implies that refinement cannot be treated as an operator; hence the standard congruence problem of traditional action refinement disappears.
- We integrate action refinement with interleaving semantics. To our knowledge, the only other works that are even remotely similar are [10], which studies the traditional congruence problem for action refinement with the aim of establishing restrictions under which interleaving models are still compositional, and [22], which considers a different type of action refinement where the refinements are explicitly serialized—an operation for which interleaving models are in fact already compositional.
- We directly compare systems on different levels of abstraction, using a concept of *vertical implementation relation* that extends the standard notion of "horizontal" implementation relation.
- We give algebraic proof rules for vertical implementation. The only comparable concept in traditional action refinement seems to be its treatment as syntactic substitution, studied by Aceto and Hennessy in [1, 2] and compared by us with semantic refinement in [21].
- We allow vertical implementation to be *collapsed* to the well-known rooted bisimilarity relation, by hiding all the actions that were refined, reminiscent of the interface refinement principle discussed in [7]. This makes it possible to mix action refinement with established methods for "horizontal" implementation.

Some of the basic ideas behind the approach of this paper were proposed first (in a restrictive setting) in [22] and later (independently) in [33, 34]. However, the technical material, including the algebraic proof rules and the notion of vertical bisimulation, is completely new in this approach.

Vertical Composition. Our proof theory is subject to improvement. For instance, one may wish to consider the following additional rule concerning the composition of vertical refinement steps,

$$\frac{\Gamma \vdash t \sqsubseteq^{r_1} u,\ u \sqsubseteq^{r_2} v}{\Gamma \vdash t \sqsubseteq^{r_2 \circ r_1} v} \mathrm{R}_{27},$$

where $r_2 \circ r_1$ is the composition of the refinement functions r_2 after r_1, for instance by $a \mapsto r_2^*(r_1(a))$. Indeed, vertical bisimulation does not satisfy this rule, for a very surprising reason: adopting the rule would reintroduce the standard congruence problem of the traditional approach to action refinement! Consider:

1. Rule R_6 implies that $a \lesssim^{a \mapsto t} t$.
2. Applying Rule R_{27}, if $t \lesssim^r u$ then $a \lesssim^{a \mapsto r^*(t)} u$.
3. Therefore, if $t \lesssim^r u_1$ and $t \lesssim^r u_2$ then $a \lesssim^{a \mapsto r^*(t)} u_i$ for $i = 1, 2$.
4. Definition 4.3 implies that $a \lesssim^r t$ if *and only if* $t \simeq r(a)$.
5. Combining steps 3 and 4, if $t \lesssim^r u_1$ and $t \lesssim^r u_2$ then $u_1 \simeq u_2$.
6. Applying also Rule R_3, if $t_1 \simeq t_2$, $t_1 \lesssim^r u_1$ and $t_2 \lesssim^r u_2$ then $u_1 \simeq u_2$.

Since among other things, $t \lesssim^r r^*(t)$ for all distinct r, and we know well enough that rooted bisimilarity is *not* a congruence for syntactic action refinement, $\lesssim^r$ cannot satisfy Rule R_{27}.

The above line of reasoning is quite generic; crucial points seem to be the definition of composition of refinement functions and step 4. It can be concluded that if a vertical implementation relation $\lesssim^r$ is based on an interleaving relation $\cong$ and satisfies both Rules R_3 and R_{27}, then either refinement function composition must be defined in some other way, or $a \lesssim^r t$ may not automatically imply that $t \cong r(a)$. In particular, if $\cong$ is rooted bisimilarity as in this paper, a notion of vertical bisimulation satisfying Rule R_{27} must be *weaker* than $\lesssim^r$. On the other hand, we have found that weaker versions of $\lesssim^r$ satisfying R_{27} may easily fail to satisfy R_{22} and R_{23}, so that "solution" would be worse than the problem.

Lax Refinement. A problem in the context of action refinement that we have touched upon several times is that traditional refinement is too *strict*: it forces all abstract causalities to be inherited in the implementation. To some degree, we have solved this problem by "closing up to rooted bisimilarity," so that apparent abstract causalities may sometimes be turned into independencies (as in $a; b + b; a \lesssim^{a \mapsto a_1; a_2} a_1; a_2 \,|||\, b$), and by formulating R_{24}, which states that activities that on an abstract level were specified completely after a may in the implementation overlap the "tail" of the refinement of a. Examples of this rule can be found in Section 7. The following rule is yet more permissive:

$$\frac{r(a) = u_1; u_2 \quad \Gamma \vdash t \sqsubseteq^r v_1; v_2}{\Gamma \vdash a; t \sqsubseteq^r u_1; (u_2 \,|||\, v_1); v_2}.$$

This expresses that *any initial fragment* of the implementation of t may overlap with the tail of the refinement of a; it does not need to be the entire implementation of t. The reason we have not put this rule in the desiderata for vertical implementation, rather than R_{24}, is that it is not sound for $\lesssim^r$ (for instance, $a; b \not\lesssim^r a_1; (a_2 \,|||\, b_1); b_2$ if $r: a \mapsto a_1; a_2, b \mapsto b_1; b_2$).

A possible "relaxation" of another kind concerns choice rather than sequential composition. In this paper we have required that all options specified by a refinement function must indeed be offered by the refined system. For instance, up to $r: a \mapsto a'; b + a'; c$, the abstract system $a; d$ is implemented by the concrete system $(a'; b + a'; c); d$. An interesting alternative is to take the decision about which option to implement during the refinement step, hence allowing $a'; b; d$ or $a'; c; d$ as an implementation, or to turn the nondeterministic choice into a deterministic one, hence allowing $a'; (b + c); d$ as an implementation. For instance, in the booking agent example, it would be more reasonable to let the customer decide whether he wants his booking info to be printed directly or mailed to him, instead of having him accept

both possibilities (as it is now). Again, this would be reflected by additional derivation rules; for instance, the first alternative is expressed by a rule of the form

$$\frac{\Gamma \vdash t \sqsubseteq^{r_1} u}{\Gamma \vdash t \sqsubseteq^{r_1+r_2} u},$$

where $(r_1 + r_2)\colon a \mapsto r_1(a) + r_2(a)$ for all $a \in \mathbf{A}$ (for instance, in the example above, $r_1\colon a \mapsto a'; b$, $r_2\colon a \mapsto a'; c$ and $r = r_1 + r_2$).

In [24] we have developed a notion of "lax vertical bisimulation," satisfying the above rules; however, this in turn fails to satisfy R_{23}. Rather, to give a vertical implementation of communicating parallel subsystems, in the approach of [24] it is necessary to give a *strict* vertical implementation of one component and a *lax* vertical implementation of the other. Since, as it turns out, the soundness of R_{22} and R_{23} is quite sensitive to changes in the definition of vertical bisimulation, it may be the case that such a combination of strong and lax refinement is the best possible compromise between the different, maybe intrinsically contrasting, desiderata for vertical implementation.

Varying the Basis. We have chosen rooted bisimilarity as the basis of our vertical implementation relation because it is well known, has a well-understood theory, is easy to visualize and is straightforward to prove on finite-state systems. However, we feel that any τ-abstracting congruence (see [15] for an overview) would probably also be suitable as a basis for vertical implementation. Natural interesting candidates are *vertical testing* (see also [33]) and *branching bisimulation* (see [20]). Indeed, the latter was investigated in [24] and gives rise to a notion of vertical bisimulation that is in some ways simpler than the one proposed in this paper.

APPENDIX: PROOFS OF SELECTED THEOREMS

First, we prove the soundness of the rules in Table 3 for vertical bisimilarity of closed terms.

THEOREM 4.1. *$\lesssim^r$ satisfies all the rules in Table* 3.

Proof. R_1. The relation $\rho = \{(t, t, \emptyset) \mid t \in \mathbf{L}\}$ is trivially a weak vertical bisimulation relation up to *id*, such that syntactic equality over $\mathbf{L}$ is a biroot of $\rho^{\emptyset}$.

R_2. Note that for the identity refinement, the active domain is empty and there can be no pending refinements; i.e., $\mathrm{adom}(r) = \mathrm{arng}(r) = \emptyset$ and $rsd(id) = \{\emptyset\}$. It follows that down-simulation and up-simulation relations up to *id* are simply weak simulation relations, whereas the "residual" simulation is irrelevant. Hence, any weak vertical bisimulation ρ up to *id* gives rise to a weak bisimulation $\rho^{\emptyset}$. It automatically follows that any biroot of $\rho^{\emptyset}$ is a subrelation of $\lesssim^r$.

R_3. We show that $\rho \subseteq \mathbf{L} \times \mathbf{L} \times rsd(r)$ with $\rho^R = \approx \circ \preccurlyeq^{r,R} \circ \approx$ for all $R \in rsd(r)$ is a weak vertical bisimulation relation. It automatically follows that any biroot of $\rho^{\emptyset}$ is a subrelation of $\lesssim^r$.

Let us first show that $\rho^{\emptyset}$ is a down-simulation. Consider $t_1\ \rho^{\emptyset}\ t_4$ due to $t_1 \approx t_2 \preccurlyeq^{r,\emptyset} t_3 \approx t_4$. If $t_1 \xrightarrow{\alpha} t_1'$, then

$$t_2 \stackrel{\varepsilon}{\Rightarrow} u_2 \xrightarrow{\hat{\alpha}} u_2' \stackrel{\varepsilon}{\Rightarrow} t_2'$$

with $t_1' \approx t_2'$ (where $\xrightarrow{\hat{\alpha}}$ stands for equality if $\alpha = \tau$, and for $\xrightarrow{\alpha}$ otherwise). Due to $t_2 \preccurlyeq^{r,\emptyset} t_3$, it follows that $t_3 \stackrel{\varepsilon}{\Rightarrow} u_3$ with $u_2 \lesssim^{r,\emptyset} u_3$. We recognize two cases.

1. $\alpha \in \mathrm{adom}(r)$. For all $r(\alpha) \stackrel{\sigma\surd}{\Longrightarrow}$, it follows that $u_3 \stackrel{\sigma}{\Rightarrow} u_3'$ with $u_2' \preccurlyeq^{r,\emptyset} u_3'$; hence $u_3' \stackrel{\varepsilon}{\Rightarrow} t_3'$ with $t_2' \preccurlyeq^{r,\emptyset} t_3'$. Due to $t_3 \stackrel{\sigma}{\Rightarrow} t_3'$, also $t_4 \stackrel{\sigma}{\Rightarrow} t_4'$ with $t_3' \approx t_4'$.

2. $\alpha \notin \mathrm{adom}(r)$. If $\alpha = \tau$ then $u_2 = u_2'$ and hence $u_2' \lesssim^{r,\emptyset} u_3'$ for $u_3' = u_3$; otherwise $u_3 \stackrel{\alpha}{\Rightarrow} u_3'$ such that $u_2' \preccurlyeq^{r,\emptyset} u_3'$. In either case, it then follows that $u_3' \stackrel{\varepsilon}{\Rightarrow} t_3'$ such that $t_2' \preccurlyeq^{r,\emptyset} t_3'$. Due to $t_3 \stackrel{\hat{\alpha}}{\Rightarrow} t_3'$, also $t_4 \stackrel{\hat{\alpha}}{\Rightarrow} t_4'$ such that $t_3' \approx t_4'$.

In each case, it follows that $t_1'\ \rho^{\emptyset}\ t_4'$; hence we are done.

We now show that ρ is an up-simulation. Consider $t_1 \,\rho\, t_4$ due to $t_1 \approx t_2 \preccurlyeq^{r,R} t_3 \approx t_4$. If $t_4 \xrightarrow{\gamma} t_4'$, then

$$t_3 \stackrel{\varepsilon}{\Rightarrow} u_3 \xrightarrow{\hat{\gamma}} u_3' \stackrel{\varepsilon}{\Rightarrow} t_3',$$

such that $t_3' \approx t_4'$. Due to $t_2 \preccurlyeq^{r,R} t_3$, it follows that $t_2 \stackrel{\varepsilon}{\Rightarrow} u_2$ with $u_2 \lesssim^{r,R} u_3$. We now recognize three cases:

1. $\exists \alpha \in \mathrm{adom}(r)$: $u_2 \stackrel{\alpha}{\Rightarrow} u_2'$ and $r(\alpha) \xrightarrow{\gamma} v$ with $u_2' \lesssim^{r,R\oplus[v]} u_3'$. It then follows that $u_2' \stackrel{\varepsilon}{\Rightarrow} t_2'$ such that $t_2' \preccurlyeq^{r,R\oplus[v]} t_3'$. Due to $t_2 \stackrel{\alpha}{\Rightarrow} t_2'$, also $t_1 \stackrel{\alpha}{\Rightarrow} t_1'$ such that $t_1' \approx t_2'$. It follows that $t_1' \,\rho^{R\oplus[v]}\, t_4'$.
2. $u_2 \stackrel{\varepsilon}{\Rightarrow} u_2'$ and $R \xrightarrow{\gamma} R'$ with $u_2' \lesssim^{r,R'} u_3'$. It then follows that $u_2' \stackrel{\varepsilon}{\Rightarrow} t_2'$ such that $t_2' \preccurlyeq^{r,R'} t_3'$. Due to $t_2 \stackrel{\varepsilon}{\Rightarrow} t_2'$, also $t_1 \stackrel{\varepsilon}{\Rightarrow} t_1'$ such that $t_1' \approx t_2'$. It follows that $t_1' \rho^{R'} t_4'$.
3. $\gamma \notin \mathrm{arng}(r)$. If $\gamma = \tau$ then $u_3 = u_3'$ and hence $u_2' \lesssim^{r,R} u_3'$ for $u_2' = u_2$; otherwise $u_2 \stackrel{\gamma}{\Rightarrow} u_2'$ such that $u_2' \preccurlyeq^{r,R} u_3'$. In either case, it then follows that $u_2' \stackrel{\varepsilon}{\Rightarrow} t_2'$ such that $t_2' \preccurlyeq^{r,R} t_3'$. Due to $t_2 \stackrel{\hat{\gamma}}{\Rightarrow} t_2'$, also $t_1 \stackrel{\hat{\gamma}}{\Rightarrow} t_1'$ such that $t_1' \approx t_2'$. It follows that $t_1' \,\rho^R\, t_4'$.

Finally, we show that for all $t \in \mathbf{L}$, the relation $\kappa = \{(R, u) \mid t \,\rho^R\, u\}$ is a "residual" weak simulation. This is due to the fact that $\kappa = \kappa' \circ \approx$ where $\kappa' = \bigcup_{t_0 \approx t} \{(R, u_0) \mid t_0 \preccurlyeq^{r,R} u_0\}$; here, the relations $\{(R, u_0) \mid t_0 \preccurlyeq^{r,R} u_0\}$ (for arbitrary t_0) and $\approx$ are weak simulations, and union and composition of weak simulations yield weak simulations.

R_4. The relation $\rho = \{(\mathbf{0}, \mathbf{0}, \emptyset)\}$ is a weak vertical bisimulation relation (for any r), and $\{(\mathbf{0}, \mathbf{0})\}$ is a biroot of $\rho^\emptyset$.

R_5. The relation $\rho = \{(\mathbf{1}, \mathbf{1}, \emptyset), (\mathbf{0}, \mathbf{0}, \emptyset)\}$ is a weak vertical bisimulation relation (for any r), and $\{(\mathbf{1}, \mathbf{1})\}$ is a biroot of $\rho^\emptyset$.

R_6. If $\alpha \notin \mathrm{adom}(r)$ then the statement is obvious. Otherwise,

$$\{(\alpha, r(\alpha), \emptyset), (\mathbf{0}, \mathbf{0}, \emptyset)\} \cup \left\{(\mathbf{1}, t, [t]) \,\middle|\, \exists \sigma \in \mathbf{U}^+ : r(\alpha) \stackrel{\sigma}{\Rightarrow} t\right\}$$

is a weak vertical bisimulation relation up to r, and $\{(\alpha, r(\alpha))\}$ is a biroot of $\rho^\emptyset$.

R_7. The relation $\rho = \{(t_1 + t_2, u_1 + u_2, \emptyset) \mid t_1 \lesssim^r u_1, t_2 \lesssim^r u_2\} \cup \preccurlyeq^r$ is a weak vertical bisimulation relation up to r, and $\{(t_1 + t_2, u_1 + u_2) \mid t_1 \lesssim^r u_1, t_2 \lesssim^r u_2\}$ is a biroot of $\rho^\emptyset$.

R_8. The relation $\rho = \{(t_1; t_2, u_1; u_2, R) \mid t_1 \preccurlyeq^{r,R} u_1, t_2 \lesssim^r u_2\} \cup \preccurlyeq^r$ is a weak vertical bisimulation relation up to r, and $\{(t_1; t_2, u_1; u_2) \mid t_1 \lesssim^r u_1, t_2 \lesssim^r u_2\}$ is a biroot of $\rho^\emptyset$.

R_9. Assume $\phi \restriction \mathrm{adom}(r) = id_{\mathrm{adom}(r)}$. The relation $\rho = \{(t[\phi], u[\phi], R[\phi]) \mid t \preccurlyeq^{r,R} u\}$, where $R[\phi] = \sum_{v \in R} v[\phi]$ for all $R \in rsd(r)$, is then a weak vertical bisimulation relation up to r, and $\{(t[\phi], u[\phi]) \mid t \lesssim^r u\}$ is a biroot of $\rho^\emptyset$.

R_{10}. Assume that r preserves A, and let $C = \mathcal{A}(r(A))$. For arbitrary $R \in rsd(r)$, let

$$R \backslash C = \{v \in R \mid \mathcal{A}(v) \cap C = \emptyset\}.$$

We first prove that $\rho = \{(t/A, u/C, R\backslash C) \mid t \preccurlyeq^{r,R} u\}$ is a weak vertical bisimulation relation up to $r\backslash A$.

First, we show that $\rho^\emptyset$ is a down-simulation. Assume $t/A \,\rho^\emptyset\, u/C$; hence $t \preccurlyeq^{r,R} u$ such that $R\backslash C = \emptyset$. Moreover, assume that $t/A \xrightarrow{\alpha} t'/A$.

Since r preserves A, it follows that $\mathcal{A}(v) \subseteq C$ for all $v \in R$. Since, moreover, all $v \in R$ are finite and terminating terms (due to the definition of $\mathbf{R}$), $R \stackrel{\sigma}{\Rightarrow} \emptyset$ for some $\sigma \in C^*$. By the fact that $\{(R, u) \mid t \preccurlyeq^{r,R} u\}$ is a weak simulation, it follows that $u \stackrel{\sigma}{\Rightarrow} u'$ such that $t \preccurlyeq^{r,\emptyset} u'$. We now recognize two cases.

1. $\alpha \in \mathrm{adom}(r)$; hence $\alpha \notin A$ and $t \xrightarrow{\alpha} t'$. For all $(r\backslash A)(\alpha) \stackrel{\sigma\surd}{\Longrightarrow}$, also $r(\alpha) \stackrel{\sigma\surd}{\Longrightarrow}$; hence $u' \stackrel{\sigma}{\Rightarrow} u''$ such that $t' \preccurlyeq^{r,\emptyset} u''$. It follows that $\sigma \in (\mathbf{C}\backslash C)^*$ and hence $u'/C \stackrel{\sigma}{\Rightarrow} u''/C$ and $t'/A \,\rho^\emptyset\, u''/C$.
2. $\alpha \notin \mathrm{adom}(r)$. There are two subcases.

—$\alpha = \tau$ and $t \xrightarrow{\beta} t'$ for some $\beta \in A$. Let $r(\beta) \stackrel{\sigma\surd}{\Longrightarrow}$ (such a transition always exists due to $r(\beta) \in \mathbf{R}$); then $u' \stackrel{\sigma}{\Rightarrow} u''$ such that $t' \preccurlyeq^{r,\emptyset} u''$. Since $\sigma \in C^*$, it follows that $u'/C \stackrel{\varepsilon}{\Rightarrow} u''/C$ and $t'/A\, \rho^{\emptyset}\, u''/C$.

—$t \xrightarrow{\alpha} t'$. Then $u' \stackrel{\hat{\alpha}}{\Rightarrow} u''$ such that $t' \preccurlyeq^{r,\emptyset} u''$; since $\alpha \notin A$, it follows that $u'/C \stackrel{\hat{\alpha}}{\Rightarrow} u''/C$ such that $t'/A\, \rho^{\emptyset}\, u''/C$.

We now show that ρ is an up-simulation. Assume $t/A\, \rho^R\, u/C$; hence $t \preccurlyeq^{r,R_0} u$ such that $R_0 \backslash C = R$. Moreover, assume that $u/C \xrightarrow{\gamma} u'/C$. We recognize two cases.

—$u \xrightarrow{\gamma} u'$ and $\gamma \notin C$. There are three subcases.

1. $\exists \alpha \in \mathrm{adom}(r)$: $t \stackrel{\alpha}{\Rightarrow} t'$ and $r(\alpha) \xrightarrow{\gamma} v$ such that $t' \preccurlyeq^{r,R_0 \oplus [v]} u'$. It follows that $\alpha \in \mathrm{adom}(r \backslash A)$ and $\mathcal{A}(v) \cap C = \emptyset$, implying that $(R_0 \oplus [v]) \backslash C = R \oplus [v]$; hence $t/A \stackrel{\alpha}{\Rightarrow} t'/A$ and $(r \backslash A)(\alpha) \xrightarrow{\gamma} v$ such that $t'/A\, \rho^{R \oplus [v]}\, u'/C$.

2. $t \stackrel{\varepsilon}{\Rightarrow} t'$ and $R_0 \xrightarrow{\gamma} R_0'$ such that $t' \preccurlyeq^{r,R_0'} u'$. It follows that $R_0' = R_0 \ominus [v] \oplus [v']$ such that $v \xrightarrow{\gamma} v'$; since $\gamma \notin C$ and r preserves A, it follows that $\mathcal{A}(v) \cap C = \emptyset$. Hence $v \in R$ and $R \xrightarrow{\gamma} R' = R \ominus [v] \oplus [v']$, where, moreover, $R' = R_0' \backslash C$. We may conclude that $t/A \stackrel{\varepsilon}{\Rightarrow} t'/A$ and $t'/A\, \rho^{R'}\, u'/C$.

3. $\gamma \notin \mathrm{arng}(r)$ and $t \stackrel{\hat{\gamma}}{\Rightarrow} t'$ such that $t' \preccurlyeq^{r,R_0} u'$. It follows that $\gamma \notin \mathrm{arng}(r \backslash A)$; moreover, $t/A \stackrel{\hat{\gamma}}{\Rightarrow} t'/A$ and $t'/A\, \rho^R\, u'/C$.

—$\gamma = \tau$ and $u \xrightarrow{\delta} u'$ for some $\delta \in C$. It follows that $\gamma \notin \mathrm{arng}(r \backslash A)$. Again, there are three subcases.

1. $\exists \alpha \in \mathrm{adom}(r)$: $t \stackrel{\alpha}{\Rightarrow} t'$ and $r(\alpha) \xrightarrow{\delta} v$ such that $t' \preccurlyeq^{r,R_0 \oplus [v]} u'$. Since r preserves A, it follows that $\alpha \in A$ and $\mathcal{A}(v) \subseteq C$; hence $(R_0' \oplus [v]) \backslash C = R$, implying that $t/A \stackrel{\varepsilon}{\Rightarrow} t'/A$ and $t'/A\, \rho^R\, u'/C$.

2. $t \stackrel{\varepsilon}{\Rightarrow} t'$ and $R_0 \xrightarrow{\delta} R_0'$ such that $t' \preccurlyeq^{r,R_0'} u'$. It follows that $R_0' = R_0 \ominus [v] \oplus [v']$ such that $v \xrightarrow{\delta} v'$; since $\delta \in C$ and r preserves A, this implies that $\mathcal{A}(v) \subseteq C$ and $\mathcal{A}(v') \subseteq C$, and hence $R_0' \backslash C = R_0 \backslash C$. We may conclude that $t'/A\, \rho^R\, u'/C$.

3. $\delta \notin \mathrm{arng}(r)$ and $t \stackrel{\hat{\delta}}{\Rightarrow} t'$ such that $t' \preccurlyeq^{r,R_0} u'$. It follows that $\delta \in A$; hence $t/A \stackrel{\varepsilon}{\Rightarrow} t'/A$ and $t'/A\, \rho^R\, u'/C$.

Finally, we show that for all t/A, $\kappa = \{(R, u/C) \mid t/A\, \rho^R\, u/C\}$ is a weak simulation. Assume $R\, \kappa\, u/C$; it follows that $R = R_0 \backslash C$ where $t \preccurlyeq^{r,R_0} u$. Now assume that $R \xrightarrow{\gamma} R'$. It follows that $\gamma \notin C$ and $R' = R \ominus [v] \oplus [v']$ such that $v \xrightarrow{\gamma} v'$ and $\mathcal{A}(v) \cap C = \mathcal{A}(v') \cap C = \emptyset$; hence $R_0 \xrightarrow{\gamma} R_0' \ominus [v] \oplus [v']$, where $R' = R_0' \backslash C$. This implies that $u \stackrel{\gamma}{\Rightarrow} u'$ such that $t \preccurlyeq^{r,R_0'} u'$; we may conclude $u/C \stackrel{\gamma}{\Rightarrow} u'/C$ such that $R'\, \kappa\, u'/C$.

It is straightforward to show that $\{(t/A, u/C) \mid t \lesssim^r u\}$ is a biroot of $\rho^{\emptyset}$.

R_{11}. Assume that r is distinct on A (hence r also preserves A), and let $C = \mathcal{A}(r(A))$. For arbitrary $R \in rsd(r)$, let

$$R \restriction C = \{v \in R \mid \mathcal{A}(v) \subseteq C\}$$
$$R \backslash C = \{v \in R \mid \mathcal{A}(v) \cap C = \emptyset\}.$$

Since r preserves A, $R = (R \restriction C) \oplus (R \backslash C)$ for all $R \in rsd(r)$. We now prove that

$$\rho = \left\{(t_1 \parallel_A t_2, u_1 \parallel_C u_2, (R_1 \backslash C) \oplus R_2) \,\middle|\, t_1 \preccurlyeq^{r,R_1} u_1,\ t_2 \preccurlyeq^{r,R_2} u_2,\ R_1 \restriction C = R_2 \restriction C\right\}$$

is a vertical bisimulation relation up to r.

First we prove that $\rho^{\emptyset}$ is a down-simulation. Assume that $t_i \preccurlyeq^{r,\emptyset} u_i$ for $i = 1, 2$ and $t_1 \parallel_A t_2 \xrightarrow{\alpha} t_1' \parallel_A t_2'$. We recognize three cases.

—$\alpha \notin A$, $t_1 \xrightarrow{\alpha} t_1'$ and $t_2 = t_2'$. There are two subcases.

1. $\alpha \in \mathrm{adom}(r)$, and if $r(\alpha) \stackrel{\sigma\surd}{\Longrightarrow}$, then $u_1 \stackrel{\sigma}{\Rightarrow} u_1'$ such that $t_1' \preccurlyeq^{r,\emptyset} u_1'$. Since r preserves A, we have $\sigma \in (\mathbf{C} \backslash C)^*$ and hence $u_1 \parallel_C u_2 \stackrel{\sigma}{\Rightarrow} u_1' \parallel_C u_2'$ with $u_2 = u_2'$ and $t_1' \parallel_A t_2'\, \rho^{\emptyset}\, u_1' \parallel_C u_2'$.

2. $\alpha \notin \mathrm{adom}(r)$ and $u_1 \stackrel{\hat{\alpha}}{\Rightarrow} u_1'$ such that $t_1' \preccurlyeq^{r,\emptyset} u_1'$. It follows that $u_1 \parallel_C u_2 \stackrel{\hat{\alpha}}{\Rightarrow} u_1' \parallel_C u_2'$ with $u_2 = u_2'$ and $t_1' \parallel_A t_2' \rho^{\emptyset} u_1' \parallel_C u_2'$.

—$\alpha \notin A$, $t_1 = t_1'$ and $t_2 \stackrel{\alpha}{\rightarrow} t_2'$. Symmetrical to the case above.

—$\alpha \in A$ and $t_i \stackrel{\alpha}{\rightarrow} t_i'$ for $i = 1, 2$. Again, there are two subcases.

1. $\alpha \in \mathrm{adom}(r)$, and if $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$ then $u_i \stackrel{\sigma}{\Rightarrow} u_i'$ for $i = 1, 2$ such that $t_i' \preccurlyeq^{r,\emptyset} u_i'$. Since r preserves A, we have $\sigma \in C^*$; hence $u_1 \parallel_C u_2 \stackrel{\sigma}{\Rightarrow} u_1' \parallel_C u_2'$ and $t_1' \parallel_A t_2' \rho^{\emptyset} u_1' \parallel_C u_2'$.

2. $\alpha \notin \mathrm{adom}(r)$ and $u_i \stackrel{\hat{\alpha}}{\Rightarrow} u_i'$ for $i = 1, 2$ such that $t_i' \preccurlyeq^{r,\emptyset} u_i'$. It follows that $u_1 \parallel_C u_2 \stackrel{\hat{\alpha}}{\Rightarrow} u_1' \parallel_C u_2'$ and $t_1' \parallel_A t_2' \rho^{\emptyset} u_1' \parallel_C u_2'$.

We now prove that ρ is an up-simulation. Assume $t_1 \parallel_A t_2 \rho^R u_1 \parallel_C u_2$; for $i = 1, 2$, let R_i be such that $t_i \preccurlyeq^{r,R_i} u_i$ with $R_1 \restriction C = R_2 \restriction C$ and $R = (R_1 \backslash C) \oplus R_2$. Now assume that $u_1 \parallel_A u_2 \stackrel{\gamma}{\rightarrow} u_1' \parallel_A u_2'$. We recognize three cases.

—$\gamma \notin C$, $u_1 \stackrel{\gamma}{\rightarrow} u_1'$ and $u_2 = u_2'$. We recognize three further cases.

1. $\exists \alpha \in \mathrm{adom}(r)$: $t_1 \stackrel{\alpha}{\Rightarrow} t_1'$ and $r(\alpha) \stackrel{\gamma}{\rightarrow} v$ such that $t_1' \preccurlyeq^{r,R_1 \oplus [v]} u_1'$. Since r preserves A, it follows that $\alpha \notin A$; moreover, due to $\gamma \notin C$, it follows that $(R_1 \oplus [v]) \restriction C = R_1 \restriction C = R_2 \restriction C$ and $((R_1 \oplus [v]) \backslash C) \oplus R_2 = R \oplus [v]$. We may conclude that $t_1 \parallel_A t_2 \stackrel{\alpha}{\Rightarrow} t_1' \parallel_A t_2'$ with $t_2 = t_2'$ and $t_1' \parallel_A t_2' \rho^{R \oplus [v]} u_1' \parallel_C u_2'$.

2. $t_1 \stackrel{\varepsilon}{\Rightarrow} t_1'$ and $R_1 \stackrel{\gamma}{\rightarrow} R_1'$ such that $t_1' \preccurlyeq^{r,R_1'} u_1'$. Since r preserves A and $\gamma \notin C$, it follows that $R_1' \restriction C = R_1 \restriction C = R_2 \restriction C$; let $R' = (R_1' \backslash C) \oplus R_2$. It follows that $R \stackrel{\gamma}{\rightarrow} R'$ and $t_1 \parallel_A t_2 \stackrel{\varepsilon}{\Rightarrow} t_1' \parallel_A t_2'$ with $t_2 = t_2'$, such that $t_1' \parallel_A t_2' \rho^{R'} u_1' \parallel_C u_2'$.

3. $\gamma \notin \mathrm{arng}(r)$ and $t_1 \stackrel{\hat{\gamma}}{\Rightarrow} t_1'$ such that $t_1' \preccurlyeq^{r,R_1} u_1'$. It follows that $t_1 \parallel_A t_2 \stackrel{\hat{\gamma}}{\Rightarrow} t_1' \parallel_A t_2'$ with $t_2 = t_2'$, such that $t_1' \parallel_A t_2' \rho^{R} u_1' \parallel_C u_2'$.

—$\gamma \notin C$, $u_1 = u_1'$ and $u_2 \stackrel{\gamma}{\rightarrow} u_2'$. Symmetrical to the above case (note that $R = R_1 \oplus (R_2 \backslash C)$.

—$\gamma \in C$ and $u_i \stackrel{\gamma}{\rightarrow} u_i'$ for $i = 1, 2$. We recognize three further cases.

1. $\exists \alpha \in \mathrm{adom}(r)$: $t_1 \stackrel{\alpha}{\Rightarrow} t_1'$ and $r(\alpha) \stackrel{\gamma}{\rightarrow} v$ such that $t_1' \preccurlyeq^{r,R_1 \oplus [v]} u_1'$. Since r is distinct on A, it follows that $R_2 \not\stackrel{\gamma}{\rightarrow}$ and $r(\alpha') \stackrel{\gamma}{\rightarrow} v'$ implies that $\alpha = \alpha'$ and $v = v'$; hence also $t_2 \stackrel{\alpha}{\Rightarrow} t_2'$ such that $t_2' \lesssim^{r,R_2 \oplus [v]} u_2'$. Moreover, $\alpha \in A$.

Due to $\gamma \in C$, we have $(R_1 \oplus [v]) \restriction C = (R_1 \restriction C) \oplus [v] = (R_2 \restriction C) \oplus [v] = (R_2 \oplus [v]) \restriction C$ and $((R_1 \oplus [v]) \backslash C) \oplus R_2 \oplus [v] = R \oplus [v]$. We may conclude that $t_1 \parallel_A t_2 \stackrel{\alpha}{\Rightarrow} t_1' \parallel_A t_2'$ and $t_1' \parallel_A t_2' \rho^{R \oplus [v]} u_1' \parallel_C u_2'$.

2. $t_1 \stackrel{\varepsilon}{\Rightarrow} t_1'$ and $R_1 \stackrel{\gamma}{\rightarrow} R_1'$ such that $t_1' \preccurlyeq^{r,R_1'} u_1'$. Since r is distinct on A, it follows that $r(\alpha) \not\stackrel{\gamma}{\rightarrow}$ for all $\alpha \in \mathbf{A}$; hence also $t_2 \stackrel{\varepsilon}{\Rightarrow} t_2'$ and $R_2 \stackrel{\gamma}{\rightarrow} R_2'$ such that $t_2' \preccurlyeq^{r,R_2'} u_2'$.

By definition, $R_i' = R_i \ominus [v_i] \oplus [v_i']$ for $i = 1, 2$, where $v_i \stackrel{\gamma}{\rightarrow} v_i'$. Again since r is distinct on A, and $R_i \in rsd(r)$ (for $i = 1, 2$) implies that $r(\alpha_i) \stackrel{\sigma_i}{\Rightarrow} v_i$ for some $\alpha_i \in \mathbf{A}$ and $\sigma_i \in \mathbf{C}^+$, it follows that $v_1 = v_2$ and $v_1' = v_2'$. We may conclude that $R_2' \restriction C = R_1' \restriction C$ and $R \stackrel{\gamma}{\rightarrow} R' = R \ominus [v_1] \oplus [v_1'] = (R_1' \backslash C) \oplus R_2'$. We may conclude that $t_1 \parallel_A t_2 \stackrel{\varepsilon}{\Rightarrow} t_1' \parallel_A t_2'$ and $t_1' \parallel_A t_2' \rho^{R \oplus [v_1]} u_1' \parallel_C u_2'$.

3. $\gamma \notin \mathrm{arng}(r)$ and $t_1 \stackrel{\hat{\gamma}}{\Rightarrow} t_1'$ such that $t_1' \preccurlyeq^{r,R_1} u_1'$. It follows that also $t_2 \stackrel{\hat{\gamma}}{\Rightarrow} t_2'$ such that $t_2' \lesssim^{r,R_2} u_2'$. We may conclude $t_1 \parallel_A t_2 \stackrel{\hat{\gamma}}{\Rightarrow} t_1' \parallel_A t_2'$ and $t_1' \parallel_A t_2' \rho^{R} u_1' \parallel_C u_2'$.

Finally, we prove that for arbitrary $t = t_1 \parallel_A t_2$, $\kappa = \{(R, u) \mid t \rho^R u\}$ is a weak simulation. Assume that $R \kappa u$; then $u = u_1 \parallel_C u_2$ and $R = (R_1 \backslash C) \oplus R_2$ with $R_1 \restriction C = R_2 \restriction C$ and $t_i \preccurlyeq^{r,R_i} u_i$ for $i = 1, 2$. Moreover, assume that $R \stackrel{\gamma}{\rightarrow} R'$. It follows that $R' = R \ominus [v] \oplus [v']$ such that $v \stackrel{\gamma}{\rightarrow} v'$. We recognize two cases.

—$\gamma \in C$; hence (since r preserves A) $\mathcal{A}(v) \subseteq C$. It follows that $v \in R_1 \restriction C = R_2 \restriction C$, and hence $R_i \stackrel{\gamma}{\rightarrow} R_i' = R_i \ominus [v] \oplus [v']$ for $i = 1, 2$, implying that $u_i \stackrel{\gamma}{\Rightarrow} u_i'$ such that $t_i \preccurlyeq^{R_i'} u_i'$. Moreover, $R_1' \restriction C = R_2' \restriction C$ and $R' = (R_1' \backslash C) \oplus R_2'$; hence $u_1 \parallel_C u_2 \stackrel{\gamma}{\Rightarrow} u_1' \parallel_C u_2'$ and $R' \kappa\, u_1' \parallel_C u_2'$.

—$\gamma \notin C$; hence (since r preserves A) $\mathcal{A}(v) \cap C = \emptyset$. Assume that $v \in R_1$; the case $v \in R_2$ is symmetrical. It follows that $R_1 \stackrel{\gamma}{\rightarrow} R_1' = R_1 \ominus [v] \oplus [v']$, implying that $u_1 \stackrel{\gamma}{\Rightarrow} u_1'$ such that $t_1 \preccurlyeq^{r,R_1'} u_1'$. Moreover, $R_1' \restriction C = R_1 \restriction C = R_2 \restriction C$ and $R' = (R_1' \backslash C) \oplus R_2$; hence $u_1 \parallel_C u_2 \stackrel{\gamma}{\Rightarrow} u_1' \parallel_C u_2'$ with $u_2 = u_2'$ and $R' \kappa\, u_1' \parallel_C u_2'$.

A straightforward proof shows that $\{(t_1 \parallel_A t_2, u_1 \parallel_C u_2) \mid t_1 \lesssim^r u_1, t_2 \lesssim^r u_2\}$ is a biroot of $\rho^{\emptyset}$.

R$_{12}$. Assume that $r: \mathbf{A} \to \mathbf{R_C}$ with $r(a) = u_1; u_2$. We show that the relation

$$\begin{aligned}\rho = &\{(a; t, u_1; (u_2 \parallel\!\parallel\!\parallel v), \emptyset)\}\\ &\cup \{(\mathbf{1}; t, u; (u_2 \parallel\!\parallel\!\parallel v), R) \mid \mathbf{1} \preccurlyeq^{r,R} u; u_2,\ t \lesssim^r v\}\\ &\cup \{(\mathbf{1}; t, u \parallel\!\parallel\!\parallel v, R) \mid \mathbf{1} \preccurlyeq^{r,R} u,\ t \lesssim^r v\}\\ &\cup \left\{(t, u \parallel\!\parallel\!\parallel v, R_1 \oplus R_2) \,\middle|\, \mathbf{1} \preccurlyeq^{r,R_1} u,\ t \preccurlyeq^{r,R_2} v\right\}\end{aligned}$$

is a weak vertical bisimulation relation up to r. Only the first and second components of the above union are in fact interesting: the third and fourth follow from the proof of Rule R$_{11}$ in combination with Rule R$_3$ (namely, $\mathbf{1} \preccurlyeq^{r,R} u$ with $t \lesssim^r v$ and $\mathbf{1}; t \simeq \mathbf{1} \parallel\!\parallel\!\parallel t$ implies that $\mathbf{1}; t \preccurlyeq^{r,R} u \parallel\!\parallel\!\parallel v$, and $\mathbf{1} \preccurlyeq^{r,R_1} u$ with $t \preccurlyeq^{r,R_2} v$ and $t \simeq \mathbf{1} \parallel\!\parallel\!\parallel t$ implies that $t \preccurlyeq^{r,R_1 \oplus R_2} u \parallel\!\parallel\!\parallel v$).

First we prove that $\rho^{\emptyset}$ is a down-simulation. Since $R \neq \emptyset$ if $\mathbf{1} \preccurlyeq^{r,R} u; u_2$, there is only one interesting case:

—$a; t\ \rho^{\emptyset}\ u_1; (u_2 \parallel\!\parallel\!\parallel v)$, $a; t \xrightarrow{a} \mathbf{1}; t$ and $r(a) \stackrel{\sigma\surd}{\Longrightarrow}$. It follows that $u_1 \stackrel{\sigma_1}{\Rightarrow}$ and $u_2 \stackrel{\sigma_2}{\Rightarrow} u' \xrightarrow{\surd}$ such that $\sigma_2 \in \mathbf{C}^+$ and $\sigma = \sigma_1\sigma_2$, implying that $\mathbf{1} \simeq^{r,\emptyset} u'$; hence $u_1; (u_2 \parallel\!\parallel\!\parallel v) \stackrel{\sigma}{\Rightarrow} u' \parallel\!\parallel\!\parallel v$ and $\mathbf{1}; t\ \rho^{\emptyset}\ u' \parallel\!\parallel\!\parallel v$.

Now, we show that ρ is an up-simulation. There are several interesting cases.

—$a; t\ \rho^{\emptyset}\ u_1; (u_2 \parallel\!\parallel\!\parallel v)$ and $u_1; (u_2 \parallel\!\parallel\!\parallel v) \xrightarrow{\gamma} u'; (u_2 \parallel\!\parallel\!\parallel v)$ due to $u_1 \xrightarrow{\gamma} u'$. Note that $\mathbf{1} \preccurlyeq^{r,[u';u_2]} u'; u_2$. It follows that $a; t \xrightarrow{a} \mathbf{1}; t$, $r(a) \xrightarrow{\gamma} u'; u_2$ and $\mathbf{1}; t\ \rho^{[u';u_2]} u'; (u_2 \parallel\!\parallel\!\parallel v)$.

—$\mathbf{1}; t\ \rho^{R}\ u; (u_2 \parallel\!\parallel\!\parallel v)$ and $u; (u_2 \parallel\!\parallel\!\parallel v) \xrightarrow{\gamma} u'; (u_2 \parallel\!\parallel\!\parallel v)$ due to $u \xrightarrow{\gamma} u'$. By $\mathbf{1} \preccurlyeq^{r,R} u; u_2$, it follows that $R \xrightarrow{\gamma} R'$ such that $\mathbf{1} \preccurlyeq^{r,R'} u'; u_2$; hence $\mathbf{1}; t\ \rho^{R'}\ u'; (u_2 \parallel\!\parallel\!\parallel v)$.

—$\mathbf{1}; t\ \rho^{R}\ u; (u_2 \parallel\!\parallel\!\parallel v)$ and $u; (u_2 \parallel\!\parallel\!\parallel v) \xrightarrow{\gamma} u' \parallel\!\parallel\!\parallel v$ due to $u \xrightarrow{\surd}$ and $u_2 \xrightarrow{\gamma} u'$. By $\mathbf{1} \preccurlyeq^{r,R} u; u_2$, it follows that $R \xrightarrow{\gamma} R'$ such that $\mathbf{1} \preccurlyeq^{r,R'} u'$; it follows that $\mathbf{1}; t\ \rho^{R'}\ u' \parallel\!\parallel\!\parallel v$.

—$\mathbf{1}; t\ \rho^{R}\ u; (u_2 \parallel\!\parallel\!\parallel v)$ and $u; (u_2 \parallel\!\parallel\!\parallel v) \xrightarrow{\gamma} u_2 \parallel\!\parallel\!\parallel v'$ due to $u \xrightarrow{\surd}$ and $v \xrightarrow{\gamma} v'$. Due to $t \lesssim^r v$, there are two possibilities.

1. There is an $\alpha \in \mathrm{adom}(r)$ such that $t \stackrel{\alpha}{\Rightarrow} t'$, $r(\alpha) \xrightarrow{\gamma} v''$ and $t' \preccurlyeq^{r,[v'']} v'$. Then $\mathbf{1}; t \stackrel{\alpha}{\Rightarrow} t'$ and $t' \rho^{R \oplus [v'']} u_2 \parallel\!\parallel\!\parallel v'$.
2. $\gamma \notin \mathrm{arng}(r)$ and $t \stackrel{\gamma}{\Rightarrow} t'$ such that $t' \preccurlyeq^{r,\emptyset} v'$. Then $\mathbf{1}; t \stackrel{\alpha}{\Rightarrow} t'$ and $t'\ \rho^{R \oplus \emptyset}\ u_2 \parallel\!\parallel\!\parallel v'$.

Finally, we show that for arbitrary t, $\kappa = \{(R, u) \mid t\ \rho^R u\}$ is a weak simulation. There is only one interesting case.

—$R\ \kappa\ u; (u_2 \parallel\!\parallel\!\parallel v)$ and $R \xrightarrow{\gamma} R'$. Due to $\mathbf{1} \preccurlyeq^{r,R} u; u_2$, it follows that $u; u_2 \stackrel{\gamma}{\Rightarrow} u'$ such that $\mathbf{1} \preccurlyeq^{r,R'} u'$; hence either $u \stackrel{\gamma}{\Rightarrow} u''$ such that $u' = u''; u_2$, in which case $u; (u_2 \parallel\!\parallel\!\parallel v) \stackrel{\gamma}{\Rightarrow} u''; (u_2 \parallel\!\parallel\!\parallel v)$ and $R'\ \kappa\ u''; (u_2 \parallel\!\parallel\!\parallel v)$, or $u \xrightarrow{\surd}$ and $u_2 \xrightarrow{\gamma} u'$, in which case $u; (u_2 \parallel\!\parallel\!\parallel v) \stackrel{\gamma}{\Rightarrow} u' \parallel\!\parallel\!\parallel v$ and $R'\ \kappa\ u' \parallel\!\parallel\!\parallel v$.

Finally, it is straightforward to show that $\{(a; t, u_1; (u_2 \parallel\!\parallel\!\parallel v))\}$ is a biroot of $\rho^{\emptyset}$ (neither $a; t$ nor $u_1; (u_2 \parallel\!\parallel\!\parallel v)$ can do an initial τ-transition). ■

Furthermore, we give the proofs of the auxiliary lemmas leading up to Threorem 6.2.

LEMMA 6.1. *For all $i > 0$, the following inequalities hold*:

1. $\sim_i \circ \preccurlyeq^{r,R} \circ \sim_i \subseteq \preccurlyeq^{r,R}_{i-1}$ *for all* $R \in rsd(r)$;
2. $\sim_i \circ \lesssim^r \circ \sim_i \subseteq \lesssim^r_{i-1}$.

Proof. By induction on i.

- For $i = 1$, the result is immediate.
- Assume that the lemma has been proved for all $j < i$.
  1. Assume that $s_1 \sim_i s_2 \preccurlyeq^{r,R} s_3 \sim_i s_4$.

—If $R = \emptyset$ and $s_1 \xrightarrow{\alpha} s_1'$, then $s_2 \xrightarrow{\alpha} s_2'$ such that $s_1' \sim_{i-1} s_2'$. We then recognize the following cases.

(i) $\alpha \in \mathrm{adom}(r)$. Then $r(\alpha) \stackrel{\sigma\surd}{\Longrightarrow} v$ implies that $\exists s_2 \stackrel{\sigma}{\Rightarrow} s_3'$ such that $s_2' \preccurlyeq^{r,\emptyset} s_3'$. If $i > |\sigma|$, we may deduce that $s_4 \stackrel{\sigma}{\Rightarrow} s_4'$ such that $s_3' \sim_{i-|\sigma|} s_4'$. Since $\sim_{i-|\sigma|} \supseteq \sim_{i-1}$, by the induction hypothsis it follows that $s_1' \preccurlyeq^{r,\emptyset}_{i-|\sigma|-1} s_4'$.

(ii) $\alpha \notin \mathrm{adom}(r)$. Then $\exists s_3 \stackrel{\hat{\alpha}}{\Rightarrow} s_3'$ such that $s_2' \preccurlyeq^{r,\emptyset} s_3'$; since $i > 1$, we may deduce that $s_4 \stackrel{\hat{\alpha}}{\Rightarrow} s_4'$ such that $s_3' \sim_{i-1} s_4'$. By the induction hypothesis, it follows that $s_1' \preccurlyeq^{r,\emptyset}_{i-2} s_4'$.

—If $s_4 \stackrel{\gamma}{\rightarrow} s_4'$, then $s_3 \stackrel{\gamma}{\rightarrow} s_3'$ such that $s_3' \sim_{i-1} s_4'$. We then recognize the following cases.

(i) $\exists \alpha \in \mathrm{adom}(r)$. $s_2 \stackrel{\alpha}{\Rightarrow} s_2'$ and $\exists r(\alpha) \stackrel{\gamma}{\rightarrow} v$ such that $s_2' \preccurlyeq^{r,R\oplus[v]} s_3'$. Since $i > 1$, we may deduce that $s_1 \stackrel{\alpha}{\Rightarrow} s_1'$ such that $s_1' \sim_{i-1} s_2'$. By the induction hypothesis, it follows that $s_1' \preccurlyeq^{r,R\oplus[v]}_{i-2} s_4'$.

(ii) $\exists s_2 \stackrel{\varepsilon}{\Rightarrow} s_2'$ and $\exists R \stackrel{\gamma}{\rightarrow} R'$ such that $s_2' \preccurlyeq^{r,R'} s_3'$. Since $i > 1$, we may deduce that $s_1 \stackrel{\varepsilon}{\Rightarrow} s_1'$ such that $s_1' \sim_{i-1} s_2'$. By the induction hypothesis, it follows that $s_1' \preccurlyeq^{r,R'}_{i-2} s_4'$.

(iii) $\gamma \notin \mathrm{arng}(r)$. Then $\exists s_2 \stackrel{\hat{\gamma}}{\Rightarrow} s_2'$ such that $s_2' \preccurlyeq^{r,R} s_3'$; since $i > 1$, we may deduce that $s_1 \stackrel{\hat{\gamma}}{\Rightarrow} s_1'$ such that $s_1' \sim_{i-1} s_2'$. By the induction hypothesis, it follows that $s_1' \preccurlyeq^{r,\emptyset}_{i-2} s_4'$.

—If $R \stackrel{\gamma}{\rightarrow} R'$, then $s_3 \stackrel{\gamma}{\Rightarrow} s_3'$ such that $s_2 \preccurlyeq^{r,R'} s_3'$. Since $i > 1$, we may deduce that $s_4 \stackrel{\gamma}{\Rightarrow} s_4'$ such that $s_3' \sim_{i-1} s_4'$. Since $\sim_{i-1} \supseteq \sim_i$, by the induction hypothesis it follows that $s_1' \preccurlyeq^{r,R'}_{i-2} s_4'$.

It follows from the above observations that $s_1 \preccurlyeq^{r,R}_{i-1} s_4$.

2. Assume $s_1 \sim_i s_2 \lesssim^r s_3 \sim_i s_4$. It follows that $s_2 \preccurlyeq^{r,\emptyset} s_3$, and hence (by the above case) $s_1 \preccurlyeq^{r,\emptyset}_{i-1} s_4$. Moreover, if $s_1 \stackrel{\tau}{\rightarrow} s_1'$, then (since $i > 0$) $s_2 \stackrel{\tau}{\rightarrow} s_2'$ such that $s_1' \sim_{i-1} s_2'$. Hence $s_3 \stackrel{\tau}{\Rightarrow} s_3'$ such that $s_2' \preccurlyeq^{r,\emptyset} s_3'$. Hence (since $i > 0$) $s_4 \stackrel{\tau}{\Rightarrow} s_4'$ such that $s_3' \sim_{i-1} s_4'$. It follows by the case above that $s_1' \preccurlyeq^{r,\emptyset}_{i-2} s_4'$; hence $s_1 \lesssim^r_{i-1} s_4$. ■

LEMMA 6.2. *Let $t \in \mathbb{L}^{\mathsf{swg}}$ with $\mathrm{fv}(t) \subseteq \{x\}$. For all $i \in \mathbb{N}$, $\mu x.t \sim_i \mu^i x.t$.*

Proof. The proof proceeds in three steps.

1. First, one proves the following auxiliary result: If $\mathrm{fv}(t) \subseteq \{x\}$ and x does not occur within a hiding operator in t, then $u \sim_i v$ implies that $t\langle x/u\rangle \sim_i t\langle v/x\rangle$. For $i = 0$ this is immediate, whereas for $i > 0$ it is proved by induction on the structure of t.

2. The next step is to show that if $\mathrm{fv}(t) \subseteq \{x\}$ and x is strictly guarded in t, then $u \sim_i v$ implies that $t\langle x/u\rangle \sim_{i+1} t\langle v/x\rangle$. This can be deduced from the auxiliary result of the previous step, using Propositions 2.2.1 (observing that strict guardedness implies guardedness) and 6.3.1, plus the fact that if x is strictly guarded in t and $t \stackrel{\alpha}{\rightarrow} t'$, then x does not occur in t' in the context of a hiding operator.

3. Finally, the statement in the lemma is proved by induction on i, using the fact that the behavior of $\mu x.t$ equals that of $t\langle \mu x.t/x\rangle$. For $i = 0$, the statement is immediate, whereas otherwise it is obtained by applying the result of the previous step to the induction hypothesis. ■

Finally, we can prove the paper's main result.

THEOREM 6.3. *Rooted vertical bisimilarity satisfies all the rules in Table* 5.

Proof. The soundness proofs of R_{16}–R_{24} are straightforward extensions of the case for closed terms, since essentially nothing happens with the implementation environments.

R_{13}. Assume $f, g: \mathrm{fv}(t) \to \mathbf{L}$. It follows from the closed case (Rule R_1) that $f \lesssim^{id}_{\mathrm{fv}(t):id} g$ iff $f(x) \simeq g(x)$ for all $x \in \mathrm{fv}(t)$; hence Rule R_{13} states that $t\langle f\rangle \simeq t\langle g\rangle$ for all such pairs f, g. This is a consequence of the congruence of $\simeq$ (Proposition 2.4).

R_{14}. Assume that $t \lesssim^{id}_{\mathrm{fv}(t,u):id} u$. Let $f: \mathrm{fv}(t,u) \to \mathbf{L}$ be arbitrary; then $f \lesssim^{\mathrm{fv}(t,u):id} f$ due to Rule R_1. It follows that $t\langle f\rangle \lesssim^{id} u\langle f\rangle$, which by Rule R_2 implies that $t\langle f\rangle \simeq u\langle f\rangle$. By definition of the open term extension of $\simeq$ (see (2)), it follows that $t \simeq u$.

R_{15}. Assume that $t \simeq t' \lesssim^r_\Gamma u' \simeq u$. Let $f, g: \mathrm{dom}\,\Gamma \to \mathbf{L}$ be such that $f \lesssim^\Gamma g$; then $t\langle f\rangle \simeq t'\langle f\rangle \lesssim^r u'\langle g\rangle \simeq u\langle g\rangle$, and hence (by R_3) $t\langle f\rangle \lesssim^r u\langle g\rangle$. By definition of $\lesssim^r_\Gamma$ (see (3)), it follows that $t \lesssim^r_\Gamma u$.

R_{25}. Immediate, by definition of $\lesssim^r$ over open terms; see (3).

R_{26}. Proved in Theorem 6.2. ■

REFERENCES

1. Aceto, L., and Hennessy, M. C. B. (1993), Towards action-refinement in process algebras. *Inform. and Comput.* **103**, 204–269.
2. Aceto, L., and Hennessy, M. C. B. (1994), Adding action refinement to a finite process algebra. *Inform. and Comput.* **115**, 179–247.
3. Badouel, E., and Darondeau, P. (1991), On guarded recursion. *Theoret. Comput. Sci.* **82**, 403–408.
4. Baeten, J. C. M., and Glabbeek, R. J. van (1989), Abstraction and empty process in process algebra. *Fundamenta Inform.* **12**, 221–242.
5. Baeten, J. C. M., and Weijland, W. P. (1990), "Process Algebra," Cambridge Univ. Press, Cambridge, UK.
6. Bravetti, M., and Gorrieri, R. (1999), Deciding and axiomatizing ST bisimulation for a process algebra with recursion and action refinement, *in* "Expressiveness in Concurrency" (I. Castellani and B. Victor, Eds.), *Electronic Notes in Theoretical Computer Science*, Vol. 27. Elsevier, Amsterdam. [Full report version: UBLCS-99-1, Department of Computer Science, University of Bologna.]
7. Brinksma, E., Jonsson, B., and Orava, F. (1991), Refining interfaces of communicating systems, *in* "TAPSOFT '91, Vol. 2," (S. Abramsky and T. S. E. Maibaum, Eds.), Lecture Notes in Computer Science, Vol. 494, pp. 297–312, Springer-Verlag, Berlin.
8. Brookes, S. D., Hoare, C. A. R., and Roscoe, A. W. (1984), A theory of communicating sequential processes, *J. Assoc. Comput. Mach.* **31**, 560–599.
9. Castellano, L., De Michelis, G., and Pomello, L. (1987), Concurrency vs. interleaving: An instructive example, *Bull. Eur. Ass. Theoret. Comput. Sci.* **31**, 12–15.
10. Czaja, I., van Glabbeek, R. J., and Goltz, U. (1992), Interleaving semantics and action refinement with atomic choice, *in* "Advances in Petri Nets 1992" (G. Rozenberg, Ed.), Lecture Notes in Computer Science, Vol. 609, pp. 89–109, Springer-Verlag, Berlin.
11. De Nicola, R., and Hennessy, M. C. B. (1984), Testing equivalences for processes, *Theoret. Comput. Sci.* **34**, 83–133.
12. Degano, P., and Gorrieri, R. (1995), A causal operational semantics of action refinement, *Inform. and Comput.* **122**, 97–119.
13. Degano, P., Gorrieri, R., and Rosolini, G. (1992), A categorical view of process refinement, *in* "Semantics: Foundations and Applications" (J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds.), Lecture Notes in Computer Science, Vol. 666, pp. 138–153, Springer-Verlag, Berlin.
14. Glabbeek, R. J. van (1990), The refinement theorem for ST-bisimulation semantics, *in* "Programming Concepts and Methods," IFIP, North-Holland, Amsterdan.
15. Glabbeek, R. J. van (1993), The linear time–branching time spectrum II: The semantics of sequential systems with silent moves, *in* "Concur '93" (E. Best, Ed.), Lecture Notes in Computer Science, Vol. 715, pp. 66–81, Springer-Verlag, Berlin.
16. Glabbeek, R. J. van, and Goltz, U. (1990), Equivalences and refinement, *in* "Semantics of Systems of Concurrent Processes" (I. Guessarian, Ed.), Lecture Notes in Computer Science, Vol. 469, pp. 309–333, Springer-Verlag, Berlin.
17. Glabbeek, R. J. van, and Goltz, U. (1990), Refinement of actions in causality based models, *in* "Stepwise Refinement of Distributed Systems—Models, Formalisms, Correctness" (J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds.), Lecture Notes in Computer Science, Vol. 430, pp. 267–300, Springer-Verlag, Berlin.
18. Glabbeek, R. J. van, and Goltz, U. Refinement of actions and equivalence notions for concurrent systems, Acta Informatica, **37**(4/5), pp. 229–327, 2001.
19. Glabbeek, R. J. van, and Vaandrager, F. W. (1987), Petri net models for algebraic theories of concurrency, *in* "PARLE—Parallel Architectures and Languages Europe, Volume II: Parallel Languages" (J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, Eds.), Lecture Notes in Computer Science, Vol. 259, pp. 224–242, Springer-Verlag, Berlin.
20. Glabbeek, R. J. van, and Weijland, W. P. (1996), Branching time and abstraction in bisimulation semantics, *J. Assoc. Comput. Mach.* **43**, 555–600. [Extended abstract in Proc. IFIP Conference, 1989.]
21. Goltz, U., Gorrieri, R., and Rensink, A. (1996), Comparing syntactic and semantic action refinement, *Inform. and Comput.* **125**, 118–143.
22. Gorrieri, R. (1992), A hierarchy of system descriptions via atomic linear refinement, *Fundamenta Inform.* **16**, 289–336.
23. Gorrieri, R., and Laneve, C. (1995), Split and ST bisimulation semantics, *Inform. and Comput.* **116**, 272–288.
24. Gorrieri, R., and Rensink, A. Action refinement, *in* "Handbook of Proacess Algebra" (J. Bergstra, A. Ponse, and S. Smolka Eds.), Elsevier, Amsterdam, 2001, Chapter 16, pp. 1047–1147.
25. Hennessy, M. C. B., and Lin, H. (1995), Symbolic bisimulations, *Theoret. Comput. Sci.* **138**, 353–389.
26. Howe, D. J. (1996), Proving congruences of bisimulation in functional programming languages, *Inform. and Comput.* **124**, 103–112.
27. Huhn, M. (1996), Action refinement and property inheritance in systems of sequential agents, *in* "Concur '96: Concurrency Theory" (U. Montanari and V. Sassone, Eds.), Lecture Notes in Computer Science, Vol. 1119, pp. 639–654, Springer-Verlag, Berlin.
28. Janssen, W., Poel, M., and Zwiers, J. (1991), Action systems and action refinement in the development of parallel systems, *in* "Concur '91" (J. C. M. Baeten and J. F. Groote, Eds.), Lecture Notes in Computer Science, Vol. 527, pp. 298–316, Springer-Verlag, Berlin.
29. Milner, R. (1989), "Communication and Concurrency," Prentice-Hall, Englewood Cliffs, NJ.
30. Milner, R., and Sangiorgi, D. (1992), Barbed bisimulation, *in* "Automata, Languages and Programming" (W. Kuich, Ed.), Lecture Notes in Computer Science, Vol. 623, pp. 685–695, Springer-Verlag, Berlin.

31. Nielsen, M., Engberg, U., and Larsen, K. S. (1989), Fully abstract models for a process language with refinement, *in* "Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency" (J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds.), Lecture Notes in Computer Science, Vol. 354, pp. 523–549. Springer-Verlag, Berlin.
32. Olderog, E.-R. (Ed.) (1994), "Programming Concepts, Methods and Calculi," IFIP Transactions, Vol. A-56, IFIP.
33. Rensink, A. (1993), "Models and Methods for Action Refinement," Ph.D. thesis, University of Twente, Enschede, Netherlands.
34. Rensink, A. (1994), Methodological aspects of action refinement, *in* "Programming Concepts, Methods and Calculi" (E.-R. Olderog, Ed.), IFIP Transactions, Vol. A-56, pp. 227–246, IFIP.
35. Rensink, A. (1995), An event-based SOS for a language with refinement, *in* "Structures in Concurrency Theory" (J. Desel, Ed.), Workshops in Computing, pp. 294–309, Springer-Verlag, Berlin.
36. Rensink, A. (2000), Bisimilarity of open terms, *Inform. and Comput.* **156**, 345–385. [Report version: Hildesheimer Informatik-Bericht 5/97, http://www.cs.utwente.nl/~rensink/HIB97-5.ps.gz.]
37. Rensink, A., and Gorrieri, R. (1997), Action refinement as an implementation relation, *in* "TAPSOFT '97: Theory and Practice of Software Development" (M. Bidoit and M. Dauchet, Eds.), Lecture Notes in Computer Science, Vol. 1214, pp. 772–786, Springer-Verlag, Berlin. [Improved report version in [38].]
38. Rensink, A., and Gorrieri, R. (1998), Vertical bisimulation, Hildesheimer Informatik-Bericht 9/98, University of Hildesheim. [Available as http://www.cs.utwente.nl/~rensink/HIB98-9.ps.gz.]
39. Rensink, A., and Wehrheim, H. (1994), Weak sequential composition in process algebras. *in* "Concur '94: Concurrency Theory" (B. Jonsson and J. Parrow, Eds.), Lecture Notes in Computer Science, Vol. 836, pp. 226–241, Springer-Verlag, Berlin.
40. Sands, D. (1997), From SOS rules to proof principles: An operational metatheory for functional languages, *in* "24TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages," pp. 428–441, Assoc. Comput. Mach.
41. Vogler, W. (1991), Failures semantics based on interval semiwords is a congruence for refinement, *Distrib. Comput.* **4**, 139–162.
42. Vogler, W. (1993), Bisimulation and action refinement, *Theoret. Comput. Sci.* **114**, 173–200.
43. Vogler, W. (1996), The limit of Split_n-language equivalence, *Inform. and Comput.* **127**, 41–61.
44. Wehrheim, H. (1994), Parametric action refinement, *in* "Programming Concepts, Methods and Calculi" (E.-R. Olderog, Ed.), IFIP Transactions, Vol. A-56, pp. 247–266, IFIP.