HILDESHEIMER
INFORMATIK-
BERICHTE

ISSN 0941-3014

Arend Rensink
Roberto Gorrieri

Vertical Bisimulation

9/98 (June 1998)

UNIVERSITÄT
HILDESHEIM

Dieser Bericht ist
herausgegeben vom

Institut für
Informatik

Postfach 10 13 63
31113 Hildesheim

(This page intentionally left blank)

Vertical bisimulation*

Arend Rensink
Institut für Informatik, University of Hildesheim
Pmail: Postfach 101363, D–31113 Hildesheim
Email: rensink@informatik.uni-hildesheim.de

Roberto Gorrieri
Dipartimento di Scienze dell'Informazione, University of Bologna
Pmail: Mura Anteo Zamboni 7, I–40127 Bologna
Email: gorrieri@cs.unibo.it

Abstract

We investigate criteria to relate specifications and implementations belonging to conceptually different abstraction levels, and we propose *vertical bisimulation* as a candidate relation for this purpose. Vertical bisimulation is indexed by a function mapping abstract actions onto concrete processes, which lays down the basic connection between the levels. Vertical bisimulation is compatible with the standard interleaving semantics; in fact, if the refinement function is the identity, then vertical bisimulation collapses to the standard notion of rooted bisimulation. We prove that vertical bisimulation satisfies a number of congruence-like proof rules (notably a structural one for recursion) that offer a powerful, compositional proof technique to verify whether a certain process is an implementation for some specification. We give a number of small examples to demonstrate the advantages of this approach.

*This work has been partially supported by the European HCM network EXPRESS (Expressiveness of Languages for Concurrency). Preliminary results appeared in [28].

Contents

1 Introduction

There is a long tradition in defining process refinement theories (see, e.g., [7, 10, 20]), 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 realise 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 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, 11, 34, 26, 37, 38], 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 and then to 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 w.r.t. 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 $a_1; a_2; b$ as well as $a_1; (a_2 \,|||\, b)$, where in the latter an ordering at the abstract level (between a and b) has been partially forgotten at the concrete one (as a_2 and b are in parallel).

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

4. It is a generalisation of existing *horizontal* implementation relations; i.e., if the refinement function is the identity, then the vertical implementation $\leq^{id}$ should collapse to the horizontal implementation $\leq$. This has two consequences: (i) the theory of horizontal and vertical implementations 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 congruence-like proof rules that are sound w.r.t. $\leq^r$. This offers 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 developed 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, 13, 22] or by $[\![r]\!]([\![S]\!])$ (the

semantic substitution of the semantics of concrete processes $r(a)$ for actions a in the semantics of S) [11, 34, 14, 26]. 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 [8], it is clear that it is necessary to move to *non-interleaving* 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 [31, 15, 38, 39] it is shown that the coarsest congruence for the operator of action refinement contained in standard interleaving semantics is the ST semantics [35], a notion of equivalence which, roughly, considers actions as non-atomic activities split into two consecutive phases.

- 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 $r(a) = a_1;a_2$, then the only possible implementation is $r(S) = a_1;a_2;b$. Of course, this can be a serious drawback in practice, as pointed out in [19], because in general a causal relation at an abstract level can be partially forgotten at the concrete one (e.g., in the example above, if only a_1 is to be considered a cause for b then $a_1;(a_2 \,|||\, b)$ is an appropriate implementation). Some work has been recently devoted to define less rigid forms of action refinement that do allow some overlapping [12, 18, 25, 29, 40]. 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 non-equivalent implementations, *a fortiori* two equivalent specifications need not to have equivalent implementations. Hence, there is no longer a need to move to non-interleaving semantics.

The paper is organised 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 $\simeq$ [20]). Then, we present the class of refinement functions we use, which are restricted to map abstract actions to non-empty (i.e., not immediately terminating) processes that cannot deadlock.

Section 3 introduces the set of desired proof rules we would expect any vertical implementation $\leq^r$ to satisfy. We use $\sqsubseteq^r$ as the formal symbol in the proof rules for a vertical implementation relation parameterised by r, and $\sqsubseteq$ (without parameter) for the corresponding standard (horizontal) implementation relation. The rules can be divided into two 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. An interesting consequence of the proof system is that the (almost) standard notion of "action refinement as syntactic substitution" is a sound implementation technique 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 we have chosen is rooted weak bisimulation equivalence, $\simeq$, which has been widely used in the process algebra literature.
- 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, it 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 permits to prove non-trivial 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 labelled 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 allows to deduce vertical implementations of recursive, possibly infinite-state systems. Under the assumption of strict guardedness, recursion congruence is proved sound for $\lesssim^r$.

Section 7 shows several variations on an example taken from [6] 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 non-trivial facts about vertical bisimulation. Finally, in Section 8 we discuss further extensions of our work, concerning possible variations of the notion of vertical implementation relation.

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 $\mathbf{A}, \mathbf{C}, A, B, C$ (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 following 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 labelled $\checkmark$.
- α indicates the execution of the action α.
- $t + u$ indicates a choice between the behaviours described by the sub-terms t and u. The choice is decided by the first action (even if it is a $\checkmark$) that occurs from either sub-term, after which the other sub-term 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 behaviours described by t and u; A is a set of actions over which t and u synchronise. That is to say, actions from A can only be performed by both sub-terms in concert, whereas all other actions can be done by either sub-term in isolation. In addition, we use the following special case of parallel composition:

  $$t \mid\mid\mid u \quad = \quad t \parallel_\emptyset u \ .$$

- $t[\phi]$ 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*: $t\langle x \mathrel{:=} u \rangle$ 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, $t\langle x \mathrel{:=} \mu x.\, t \rangle$. 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 $\mu y.\, t\langle x \mathrel{:=} y \rangle$ are identified in all contexts.

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

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	$\{x\}$	$\begin{cases} f(x) & \text{if } x \in \mathrm{dom}(f) \\ x & \text{otherwise} \end{cases}$
$\mu x.\, t_1$	$\mathrm{fv}(t_1) \setminus \{x\}$	$\mu y.\, (t_1\langle x \to y\rangle\langle f\rangle)$ where $y \notin \mathrm{fv}(t) \cup \mathrm{dom}(f) \cup \mathrm{fv}(f)$

Table 2.1: Free variables and syntactic substitution

To formalise the notion of syntactic substitution, we first define the *free variables* of a term t, denoted $\mathrm{fv}(t)$, which are those variables that do not occur in the scope of a recursion operator; see Table 2.1. 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 $f\colon \mathbf{X} \rightharpoonup \mathbb{L}_\mathbf{A}$ or $f\colon X \to \mathbb{L}_\mathbf{A}$. Table 2.1 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 2.1: 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 $\mathrm{dom}(f) = X = \{x_1, \ldots, x_n\}$ and f maps each x_i to a term t_i, we also write f as an explicit list of substitutions: $f = (x_1 \to t_1, \ldots, x_n \to t_n)$ or $(x_i \to t_i)_{1 \le i \le n}$. Correspondingly, we write $t\langle f\rangle = t\langle x_1 \to t_1, \ldots x_n \to t_n\rangle$ or $t\langle x_i \to t_i\rangle_{1 \le i \le n}$. 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 formalise 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 in t is within a subterm $\alpha; u$ of t. The precise definition is slightly more flexible than that: for instance, x is also considered to be guarded in $(\mathbf{1}; a); x$. Furthermore, t is considered to be *well-guarded* if it contains only recursion over guarded terms. Note that this does *not* imply that all variables in t are guarded; for instance, $t = y$ is well-guarded.

2.1 Definition. First we define under what circumstances we call a variable $x \in \mathbf{X}$ *guarded* in an arbitrary (open) term.

- x is guarded in $\mathbf{0}$ and in α for all $\alpha \in \mathbf{A}_\tau$;

$$\frac{}{\mathbf{1} \xrightarrow{\checkmark} \mathbf{0}} \qquad \frac{}{\alpha \xrightarrow{\alpha} \mathbf{1}} \qquad \frac{t \xrightarrow{\alpha} t'}{t + u \xrightarrow{\alpha} t'} \qquad \frac{u \xrightarrow{\alpha} u'}{t + u \xrightarrow{\alpha} u'} \qquad \frac{t \xrightarrow{\alpha} t'}{t[\phi] \xrightarrow{\phi(\alpha)} t'[\phi]}$$

$$\frac{t \xrightarrow{\alpha} t' \quad \alpha \neq \checkmark}{t; u \xrightarrow{\alpha} t'; u} \qquad \frac{t \xrightarrow{\checkmark} t' \quad u \xrightarrow{\alpha} u'}{t; u \xrightarrow{\alpha} u'} \qquad \frac{t \xrightarrow{\alpha} t' \quad \alpha \notin A}{t/A \xrightarrow{\alpha} t'/A} \qquad \frac{t \xrightarrow{\alpha} t' \quad \alpha \in A}{t/A \xrightarrow{\tau} t'/A}$$

$$\frac{t \xrightarrow{\alpha} t' \quad \alpha \notin A_{\checkmark}}{t \parallel_A u \xrightarrow{\alpha} t' \parallel_A u} \qquad \frac{u \xrightarrow{\alpha} u' \quad \alpha \notin A_{\checkmark}}{t \parallel_A u \xrightarrow{\alpha} t \parallel_A u'} \qquad \frac{t \xrightarrow{\alpha} t' \quad u \xrightarrow{\alpha} u' \quad \alpha \in A_{\checkmark}}{t \parallel_A u \xrightarrow{\alpha} t' \parallel_A u'} \qquad \frac{t\langle x \mathrel{\rightarrowtail} \mu x.\, t\rangle \xrightarrow{\alpha} t'}{\mu x.\, t \xrightarrow{\alpha} t'}$$

Table 2.2: Transition rules.

- x is guarded in $t + u$ and $t \parallel_A u$ iff x is guarded in t and in u;
- x is guarded in $t; u$ if either $\mathrm{fv}(t) = \emptyset$ and x is guarded in u, or x is guarded in t;
- x is guarded in t/A, $t[\phi]$ and $\mu y.\, t$ iff x is guarded in t;
- x is *not* guarded in $\mathbf{1}$ or in y (for all $y \in \mathbf{X}$).

We call $t \in \mathbb{L}$ *well-guarded* if for 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.

2.2 Proposition. 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 $t\langle y \mathrel{\rightarrowtail} u\rangle$.
2. If t and u are well-guarded, then so is $t\langle y \mathrel{\rightarrowtail} u\rangle$.

The set of well-guarded terms will be denoted $\mathbb{L}^{\mathsf{wg}}_{\mathbf{A}}$ (and accordingly $\mathbf{L}^{\mathsf{wg}}_{\mathbf{A}}$); 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 labelled 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., $\checkmark$-labelled ones) that they must lead to deadlocked states:

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

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 2.2. 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 Baeten and Van Glabbeek [4]). Note that a choice may terminate even if only one of the operands terminates (cf. Baeten and Weijland [5]). Finally, $\checkmark$ cannot be hidden and must be synchronised upon.

The following proposition shows the consequences of (well-)guardedness for the operational semantics: guarded variables are not "used" in initial transitions, 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 property has the consequence that more properties can be proved by induction on the term structure.

2.3 Proposition. Let $t, u \in \mathbb{L}$ and $x \in \mathbf{X}$.

1. If x is guarded in t and $t\langle x{:=}u\rangle \xrightarrow{\alpha} t'$, then $t \xrightarrow{\alpha} t''$ such that $t' = t''\langle x{:=}u\rangle$.
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 2.2 by

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

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

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

2.3 Behavioural semantics

In an interleaving operational semantics such as the above, a widely accepted τ-abstracting equivalence relation is *rooted (weak) bisimilarity*; see [20]. 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

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

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 $t \stackrel{\varepsilon}{\Longrightarrow} t$ for any t) and $\hat{\alpha} = \alpha$ for all $\alpha \in \mathbf{U}_{\surd}$.

2.5 Definition. 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 $\exists s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2'$ 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 sub-relation $\tilde{\rho} \subseteq \rho$ such that for all $s_1\ \tilde{\rho}\ s_2$:
  - if $s_1 \xrightarrow{\tau} s_1'$ then $\exists s_2 \stackrel{\tau}{\Longrightarrow} s_2'$ 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 bi-root 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:

$$\alpha \parallel_\alpha \mathbf{1} \approx \mathbf{0} \not\approx \mathbf{1} \approx \mathbf{1} + \mathbf{1} \not\approx \mathbf{1} + \alpha \not\approx \alpha\ .$$

The following result is well-known (cf. [5, 20]):

2.6 Proposition. $\simeq$ 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 $r\colon \mathbf{A} \rightharpoonup \mathbf{R_C}$, 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 unnecessary) is largely technically motivated. At the same time, however, refinement terms satisfy a number of intuitive sensibility criteria. Indeed, the process $r(a)$ should be:

- *non-empty*, i.e., $r(a) \not\xrightarrow{\checkmark}$. This comes down to the principle that a visible abstract activity a (i.e., something the environment can synchronise on) cannot simply disappear during refinement.
- *eventually terminating*, i.e., $\exists t \stackrel{\sigma}{\Longrightarrow} t' \xrightarrow{\checkmark}$ for any term t reachable from $r(a)$. For instance, $\mu x.\, a; (x + b)$ is eventually terminating, but $\mu x.\, a; x$ is not. This criterion essentially requires that the refinement of a given action cannot "get stuck" during execution, either through deadlock or through livelock (divergence). 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.

2.7 Example. 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 $t = (a+d) \|_{a,b} b$ it is not possible that a and b synchronise; 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 synchronise over an occurrence of c, such that 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 $u = (c; a' + d) \|_{a',b',c} c; b'$ (this being the direct, syntactic refinement of t above). One possible run is $u \xrightarrow{c} a' \|_{a',b',c} b' \simeq \mathbf{0}$, 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.

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

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

2.8 Proposition. For all $t \in \mathbf{R}$, $\mathcal{A}(t) = \{a \mid \exists \sigma\colon t \stackrel{\sigma a}{\Longrightarrow}\}$.

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

2.9 Definition. Let $r\colon \mathbf{A} \to \mathbf{R_C}$ 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 $b \in \mathbf{A} \setminus A$;
- 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 sub-terms $t+u$ 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 A and the refinements of $\mathrm{dom}(r) \setminus A$. For instance, the function r in Example 2.7 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 $d \in \mathrm{dom}(r) \setminus \{a,b\}$. 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 2.7 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 follows:

$$\mathrm{arng}(r) = \bigcup_{r(a) \neq a} \mathcal{A}(r(a)) \qquad\qquad \mathrm{adom}(r) = \{a \mid r(a) \neq a\} \cup (\mathrm{arng}(r) \cap \mathrm{dom}(r))$$

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.

2.10 Example. If $\mathbf{A} = \mathbf{C} = \{a,b,c\}$ and $r\colon a \mapsto a;b,\ b \mapsto b,\ c \mapsto c$ 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.

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

Refinement function constants and operations. We use $id_{\mathbf{A}}\colon \mathbf{A} \to \mathbf{R}_{\mathbf{A}}$ 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:

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

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 $\mathrm{adom}(r\backslash A) = \mathrm{adom}(r) \setminus A$.

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

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

$$\frac{t_1 \sqsubseteq^r u_1 \quad t_2 \sqsubseteq^r u_2}{t_1; t_2 \sqsubseteq^r u_1; u_2}\,\mathrm{R}_8 \qquad \frac{t \sqsubseteq^r u \quad \phi \restriction \mathrm{adom}(r) = id_{\mathrm{adom}(r)}}{t[\phi] \sqsubseteq^r u[\phi]}\,\mathrm{R}_9 \qquad \frac{t \sqsubseteq^r u \quad r \text{ preserves } A}{t/A \sqsubseteq^{r \setminus A} u/\mathcal{A}(r(A))}\,\mathrm{R}_{10}$$

$$\frac{t_1 \sqsubseteq^r u_1 \quad t_2 \sqsubseteq^r u_2 \quad r \text{ is distinct on } A}{t_1 \parallel_A t_2 \sqsubseteq^r u_1 \parallel_{\mathcal{A}(r(A))} u_2}\,\mathrm{R}_{11}$$

Table 3.3: Proof rules for vertical implementation

3 Proof rules for vertical implementation

We now come to a central concept of this paper, namely the notion of *vertical implementation*. When we write $t \leq^r u$ we mean that t is an abstract system and u one of its possible implementations according to the (generic) *vertical implementation relation* $\leq^r$, where the correspondence between actions of t and computations of u is set via the refinement function r. Similarly, when we write $t \leq t'$ we mean that t and t' are two systems at the same abstraction level, related by the (generic) *horizontal* implementation relation $\leq$ (i.e., relating systems at the same abstraction level), which could be one of those studied in, e.g., [32].

In this Section we define a set of proof rules that any relation $\leq^r$ should satisfy; in the proof rules, we use the syntactic symbol $\sqsubseteq^r$ for $\leq^r$ and $\sqsubseteq$ for $\leq$. The list of rules, expressing natural "desiderata", is reported in Table 3.3. Some of the proof rules have side conditions on their applicability, concerning 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 3.3 range over closed terms only. The extension to open terms, including a new 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 3.3 in some detail.

- The first group of properties, consisting of rules R_1–R_3, expresses our basic assumption of working modulo the horizontal implementation relation. Rule R_1 simply states that every term implements itself as long as no refinement takes place; rule R_2 says that $\sqsubseteq^{id}$ is compatible with the horizontal implementation relation; 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 pre-order, which indeed is the standard requirement for horizontal implementation relations.

In this paper, we have chosen rooted 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., [32] 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 pre-congruence properties of $\sqsubseteq$ for the operators of $\mathbf{L}$. (In other words, the horizontal implementation relation $\sqsubseteq$ needs to be a pre-congruence, 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 synchronisation set A of the specification is refined in the implementation. We will comment later on its side conditions.

  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 rule can be shown to hold.

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

  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 similar, 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 special case of R_{10} is given by the following derived rule:

$$\frac{t \sqsubseteq^{r} u}{t/\operatorname{adom}(r) \sqsubseteq^{id} u/\operatorname{arng}(r)}$$

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

A small illustration of the rules is provided by the following example, which was already discussed in the introduction. It is based on the assumption that we are working in an interleaving model, where $a \mid\mid\mid b \sqsubseteq a; b + b; a$.

3.1 Example. Using Table 3.3, we can show that if a is split into $r(a) = a_1; a_2$ in the abstract system $S = a \mid\mid\mid b$, this can be implemented either by treating (the implementation of) a

and b as independent, resulting in $I_1 = a_1; a_2 \,|||\, b$, or by imposing an ordering, as in $I_2 = a_1; a_2; b + b; a_1; a_2$.

$$\dfrac{\dfrac{}{a \sqsubseteq^r a_1; a_2}\mathrm{R}_6 \quad \dfrac{}{b \sqsubseteq^r b}\mathrm{R}_6}{S \sqsubseteq^r I_1}\mathrm{R}_{11} \qquad \dfrac{\dfrac{\vdots}{S \sqsubseteq a; b + b; a} \quad \dfrac{\vdots}{a; b + b; a \sqsubseteq^r I_2}\mathrm{R}_8, \mathrm{R}_7 \quad \dfrac{\dfrac{}{I_2 \sqsubseteq^{id} I_2}\mathrm{R}_1}{I_2 \sqsubseteq I_2}\mathrm{R}_2}{S \sqsubseteq^r I_2}\mathrm{R}_3$$

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 3.3 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 mainly concerns the side condition of Rule R_{10} for hiding.

3.2 Example. Let $r \in \mathbf{R}_{\mathbf{A},\mathbf{C}}$ be a refinement function with active part $a \mapsto a; c$ and $b \mapsto b; c$ (where $c \notin \mathbf{A}$). Hence r does preserve neither $\{a\}$ nor $\{b\}$. Then, if we use the rules ignoring the side conditions, we would get the following derivation:

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

The left hand term on the bottom line contains no deadlock (there is just one transition of synchronisation, 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 on distinctness.

3.3 Example. Let r be a refinement function with active part $a \mapsto c; b + c; d$. The rules of Table 3.3 then allow the following derivation:

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

The left hand term on the bottom line contains no deadlock (there is just one transition of synchronisation, 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 synchronisation, 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 2.9) for the case of the alternative operator. 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 2.9).

3.4 Example. Let r be a refinement function with active part $a \mapsto c; a$ and $b \mapsto c; b$. The rules of Table 3.3 then allow the following 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 synchronising abstract actions are refined to terms having disjoint initial actions. But in fact, initial actions and later actions of a refinement image $r(a)$ should also be kept distinct, as the following example shows.

3.5 Example. Given a refinement function r which 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 2.9 for the case of sequential composition.

The identification of suitable side-conditions for rule R_{11} is related to the issue of "syntactic versus semantic action refinement" studied in [13], where suitable necessary conditions for syntactic substitution to distribute over the parallel operator (with synchronisation) have been singled out. Hence, different side-conditions for different classes of refinable terms are possible, according to the results presented in that paper.

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

Table 3.4: Syntactic refinement

3.3 Syntactic refinement

Although the derivation rules in Table 3.3 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 $r\colon \mathbf{A} \to \mathbf{R_C}$, syntactic substitution can be formalised as a partial function $r^*\colon \mathbb{L}_{\mathbf{A}} \to \mathbb{L}_{\mathbf{C}}$, defined in Table 3.4. 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.

3.6 Theorem. For all recursion-free $t \in \mathbf{L_A}$ and $r\colon \mathbf{A} \to \mathbf{R}$, 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 3.3. □

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

4 Vertical bisimulation

We now come to the definition of an actual vertical implementation relation that satisfies the proof rules of Table 3.3. This section starts by introducing the basic definition, built on rooted weak bisimulation (see Definition 2.5) chosen as the horizontal implementation relation. Then, we discuss that possible alternative definitions are not viable, as they give rise to inconsistencies. Finally, we present the main results of our vertical bisimulation relation, namely soundness of the rules in Table 3.3 (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, $T = \langle \mathbf{U}_{\tau,\checkmark}, S, \rightarrow \rangle$ is a fixed transition system, and $r\colon \mathbf{A} \to \mathbf{R_C}$ a refinement function.

4.1 Definition. 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 $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$ then $\exists s_2 \stackrel{\sigma}{\Longrightarrow} s_2'$ such that $s_1'\ \rho\ s_2'$;
2. $\alpha \notin \mathrm{adom}(r)$, and $\exists s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2'$ 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 refinement. The intermediate states of the implementation, traversed during such a complete run, are not investigated at all.

4.2 Example. Let $r\colon a \mapsto a_1; a_2, b \mapsto b, c \mapsto c$. 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 have to 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 parameterise the bisimulation.

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

$$rsd(r) = \{R\colon \mathbf{R} \to \mathbb{N} \mid \forall t \in R\colon \exists a \in \mathrm{dom}(r), \exists \sigma \in \mathbf{U}^+\colon r(a) \stackrel{\sigma}{\Longrightarrow} t \not\xrightarrow{\checkmark}\}\ .$$

We use the following constructions over $\mathbf{R} \to \mathbb{N}$:

$$\emptyset\colon \quad u \mapsto 0$$

$$
\begin{array}{rll}
[t]: & u \mapsto & \begin{cases} 1 & \text{if } u = t \text{ and } t \not\xrightarrow{\checkmark} \\ 0 & \text{otherwise} \end{cases} \\
R_1 \oplus R_2: & u \mapsto & R_1(u) + R_2(u) \\
R_1 \ominus R_2: & u \mapsto & max\{R_1(u) - R_2(u), 0\}
\end{array}
$$

The behaviour of a residual set corresponds to the synchronisation-free parallel composition of its elements. Formally:

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

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

An up-simulation has to 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.[1].

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 $\{(s_1, s_2) \mid (s_1, s_2, R) \in \rho\}$.

4.3 Definition. 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 $s_2 \xrightarrow{\gamma} s_2'$ then one of the following holds:

1. $\exists \alpha \in \text{adom}(r): \exists s_1 \stackrel{\alpha}{\Longrightarrow} s_1'$ and $\exists r(\alpha) \xrightarrow{\gamma} v$ such that $s_1' \ \rho^{R \oplus [v]} \ s_2'$.
2. $\exists s_1 \stackrel{\varepsilon}{\Longrightarrow} s_1'$ and $\exists R \xrightarrow{\gamma} R'$ such that $s_1' \ \rho^{R'} \ s_2'$;
3. $\gamma \notin \text{arng}(r)$ and $\exists s_1 \stackrel{\hat{\gamma}}{\Longrightarrow} s_1'$ 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 silently move. We will come back later on this issue.

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

4.4 Example. 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

$$
\begin{array}{rcl}
\rho_1 & = & \{(S, I), (\mathbf{1}, \mathbf{1}), (\mathbf{0}, \mathbf{0})\} \\
\rho_2 & = & \{(S, I, \emptyset), (\mathbf{1}, \mathbf{1}; \mathbf{0}, [a_2]), (\mathbf{1}, \mathbf{1}; a_2, [a_2]), (\mathbf{1}, \mathbf{1}, \emptyset), (\mathbf{0}, \mathbf{0}, \emptyset)\}
\end{array}
$$

[1]Other ternary bisimulation-based relations are, for instance, *history-preserving* bisimulation [33], and *symbolic* bisimulation [16].

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.

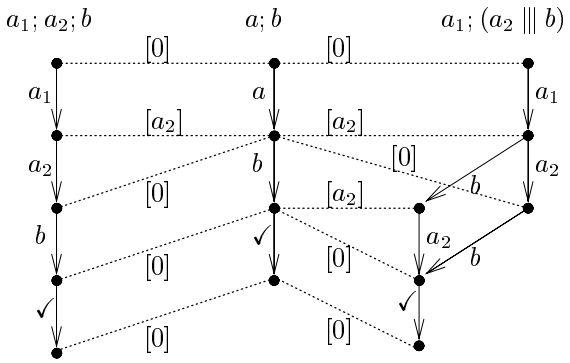
Vertical bisimulation, therefore, is determined by a relation that is both a down-simulation, an up-simulation, and a *residual simulation*, which requires that any move of the pending refinement set must be matched by the implementation, without the specification moving at all. This implies that pending 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.

4.5 Definition. 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 bi-root of $\preccurlyeq^{r,\emptyset}$.

We overload notation by re-using $\preccurlyeq^r$ to denote also the binary component $\preccurlyeq^{r,\emptyset}$. The following is an example of an actual vertical bisimulation.

4.6 Example. Let $r\colon a \mapsto a_1; a_2$. The following figure shows vertical bisimulations proving $a; b \lesssim^r a_1; a_2; b$ and $a; b \lesssim^r a_1; (a_2 \mid\mid\mid b)$, where the dotted lines pair related states and their labelling is the residual set indexing the relation:



4.2 Alternative definitions

Here we want to convince the reader, through some examples, that it is not easy to find reasonable alternatives to the definition of vertical bisimulation given above.

With respect to the down-simulation, Definition 4.5.1 requires that moves of the specification are to be matched only if there are no pending refinements, and moreover, only complete runs of $r(\alpha)$ are regarded. A stronger requirement would be to demand that down-simulation is also ternary, and satisfies the following:

1′. For all $s_1\ \rho^R\ s_2$, if $s_1 \xrightarrow{\alpha} s_1'$ then one of the following holds:

- $\alpha \in \mathrm{adom}(r)$ and if $r(\alpha) \xrightarrow{\gamma} v$ then $\exists s_2 \xrightarrow{\gamma} s_2'$ such that $s_1'\ \rho^{R\oplus[v]}\ s_2'$;
- $\alpha \notin \mathrm{adom}(r)$ and $\exists s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2'$ such that $s_1'\ \rho^R\ s_2'$

This is easily shown to be too demanding: e.g., if the specification is $a;b$ and the refinement is $r\colon a \mapsto a_1;a_2$, then this down-simulation criterion requires that in the implementation, b must be possible immediately after a_1. In other words, $a_1;a_2;b$ would not be allowed as an implementation. But even if we set $R = \emptyset$ in 1′ (so that matching is only required if there are no pending refinements), Example 4.7 below shows that this gives rise to inconsistencies.

4.7 Example. Let $r\colon a \mapsto a_1;a_2$ and consider $S_1 = a + a;(\tau + b)$, $S_2 = a;(\tau + b)$ and $I = a_1;a_2;(\tau + b)$. Clearly, $S_1 \simeq S_2$; in addition, we want $S_2 \lesssim^r I$ by Rules R_6, R_7 and R_8. But then $S_1 \lesssim^r I$ should also hold because of Rule R_3.
According to Definition 4.5, $S_1 \lesssim^r I$ indeed holds; in particular, $S_1 \xrightarrow{a} \mathbf{1}$ and $r(a) \stackrel{a_1\,a_2}{\Longrightarrow} \mathbf{1}$ can be matched by $I \xrightarrow{a_1\,a_2\,\tau} \mathbf{1}$. However, if we use criterion 1′ instead, then the only possibility is $I \xrightarrow{a_1} I' = \mathbf{1};a_2;(\tau + b)$, after which we get the pair $(\mathbf{1}, I')$ which is not in the up-simulation $\rho^{[a_2]}$, as I' can perform $b \in \mathrm{arng}(r)$ while $\mathbf{1}$ cannot.

Let us now consider the definition of up-simulation. There are two possible variations of Definition 4.3. The first one concerns Clause 1: we could require that $s_1 \xrightarrow{\alpha} s_1'$. However, this is unacceptable, as, when $r = id$ (and $R = \emptyset$), the new definition of up-simulation does no longer specialise to the standard definition of weak simulation. The second variation is about Clause 2, where we could require that the specification does not move at all, i.e., to impose $s_1 = s_1'$. However, this again turns out to be infeasible, as the following example shows.

4.8 Example. Let $r\colon a \mapsto a_1;a_2$ and consider $S = a;(\tau + b)$, $I_1 = a_1;a_2;(\tau + b)$ and $I_2 = a_1;(a_2 + a_2;(\tau + b))$. We expect $S \lesssim^r I_1$ by Rules R_6, R_7 and R_8; moreover, $I_1 \simeq I_2$ holds, which should imply $S \lesssim^r I_2$ by the compatibility of vertical implementation and rooted weak bisimulation (Rule R_3).
According to Definition 4.5, $S \lesssim^r I_2$ indeed holds; however, this is not the case with the new definition of up-simulation. In particular, $I_2 \xrightarrow{a_1} I' = \mathbf{1};(a_2 + a_2;(\tau + b))$ is matched by $S \stackrel{a}{\Longrightarrow} S' = \mathbf{1};(\tau + b)$, but then $I' \xrightarrow{a_2} \mathbf{1}$ can only be matched by a transition of the pending refinement $[a_2]$, reaching the pair $(S', \mathbf{1})$ that is not in any down-simulation $\rho^{\emptyset}$, as only S' can perform b. The only way out is to allow the specification to move during the "continuation" of this pending refinement: $S' \stackrel{\varepsilon}{\Longrightarrow} \mathbf{1}$.

Finally, we consider if the residual simulation can be modified somehow. The current criterion in Definition 4.5.3 comes down to the following:

3. For all $s_1\ \rho^R\ s_2$, if $R \xrightarrow{\gamma} R'$ then $\exists s_2 \stackrel{\gamma}{\Longrightarrow} s_2'$ such that $s_1\ \rho^{R'}\ s_2'$.

In analogy with what we discussed above, an obvious question is why the specification may not move here internally, just as in Clause 2 of up-simulation, giving rise to the following alternative:

3′. For all $s_1\ \rho^R\ s_2$, if $R \xrightarrow{\gamma} R'$ then $\exists s_1 \stackrel{\varepsilon}{\Longrightarrow} s_1'$ and $\exists s_2 \stackrel{\gamma}{\Longrightarrow} s_2'$ such that $s_1'\ \rho^{R'}\ s_2'$.

The example below shows that this would give rise to unacceptable implementations.

4.9 Example. Let $r\colon a \mapsto a_1; a_2$, and consider $S = a; (\tau + b)$ and $I = a_1; (a_2 + b; a_2) + a_1; a_2; (\tau + b)$. According to Definition 4.5, $S \not\lesssim^r I$. To see this, assume $S \; \rho^{\emptyset} \; I$ and $I \xrightarrow{a_1} \mathbf{1}; (a_2 + b; a_2) = I'$: Clause 1 of Definition 4.3 applies, with Clause 2 inapplicable since the current pending refinement set is empty. Hence we have $r(a) \xrightarrow{a_1} a_2$, resulting in the pending refinement $[a_2]$. However, S has no corresponding a-transition:

- $S \stackrel{a}{\Longrightarrow} \mathbf{1}; (\tau + b) = S'$ and $S' \; \rho^{[a_2]} \; I'$ is not satisfactory: because of the weak simulation between the pending refinement $[a_2]$ and I, move $[a_2] \xrightarrow{a_2} \emptyset$ must be matched by $I' \xrightarrow{a_2} \mathbf{1}$ where S' is not allowed to move, but $(S', \mathbf{1}) \notin \rho^{\emptyset}$ since S' can do a b;
- $S \stackrel{a}{\Longrightarrow} \mathbf{1} = S''$ and $S'' \; \rho^{[a_2]} \; I'$ is not satisfactory, since neither S'' nor $[a_2]$ can match $I' \xrightarrow{b} \mathbf{1}; a_2$.

Under the weaker Clause 3′ proposed above, however, we would have $S \lesssim^r I$; in particular, $S' \; \rho^{[a_2]} \; I'$ is now possible since $[a_2] \xrightarrow{a_2} \emptyset$ can be matched by $I' \xrightarrow{a_2} \mathbf{1}$ *combined with* $S' \stackrel{\varepsilon}{\Longrightarrow} \mathbf{1}$. However, $S \lesssim^r I$ would be inconsistent with the desired properties of vertical implementation discussed in Section 3. Consider the following derivation:

$$\dfrac{S \lesssim^r I \qquad \dfrac{\dfrac{}{a \lesssim^r a_1; a_2}\,\mathrm{R}_6 \quad \dfrac{}{c \lesssim^r c}\,\mathrm{R}_6}{a; c \lesssim^r a_1; a_2; c}\,\mathrm{R}_8}{(S \parallel_a a; c)/a \simeq (I \parallel_{a_1, a_2} a_1; a_2; c)/a_1, a_2}\,\mathrm{R}_{11}, \mathrm{R}_{10}, \mathrm{R}_2$$

The conclusion of this derivation is false, since the right hand term has the transition

$$(I \parallel_{a_1, a_2} a_1; a_2; c)/a_1, a_2 \xrightarrow{\tau} (\mathbf{1}; (a_2 + b; a_2) \parallel_{a_1, a_2} \mathbf{1}; a_2; c)/a_1, a_2 \simeq \tau; c + b; c$$

which cannot be matched by the left hand side: the only possibilities are

$$(S \parallel_a a; c)/a \stackrel{\tau}{\Longrightarrow} (\mathbf{1}; (\tau + b) \parallel_a \mathbf{1}; c)/a \simeq \tau; c + b; c + c; (\tau + b)$$

which, unlike $\tau; c + b; c$, can do cb in sequence; or

$$(S \parallel_a a; c)/a \stackrel{\tau}{\Longrightarrow} (\mathbf{1} \parallel_a \mathbf{1}; c)/a \simeq c$$

which cannot match the b-transition of $\tau; c + b; c$.

4.3 Soundness results

Directly from Definition 4.5, the following consistency result follows:

4.10 Proposition. $\preccurlyeq^{id,\emptyset} = \approx$ 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 4.3 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 3.3.

4.11 Theorem. $\lesssim^r$ satisfies all the rules in Table 3.3.

The proof is deferred to Appendix A (Page 47). The soundness result ensures that whenever $S \sqsubseteq^r I$ is provable in the proof system of Table 3.3, then $S \lesssim^r I$. As another immediate consequence, we get the following property (see Theorem 3.6):

4.12 Corollary. For all recursion-free $t \in \mathbf{L_A}$ and $r\colon \mathbf{A} \to \mathbf{R_C}$, 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.

4.13 Definition. A process $t \in \mathbf{L_A}$ is *deadlock-free* if for all t' such that $t \stackrel{\sigma}{\Longrightarrow} t'' \stackrel{\alpha}{\longrightarrow} t'$ and for all$\beta \in \mathbf{A}_{\tau,\surd}\colon t' \stackrel{\beta}{\not\longrightarrow}$, then $\alpha = \surd$.

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

4.14 Theorem. $\lesssim^r$ preserves deadlock freedom.

As a trivial corollary of Theorems 4.14 and 4.11, we have that also *provable* vertical implementation (using the proof rules of Section 3) preserves deadlock freedom.

4.15 Corollary. $\sqsubseteq^r$ 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 characterised 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 labelling: if a transition of U "opens" a new refinement, then the corresponding transition of the abstraction is labelled with the action being refined; if such transition "continues" a pending refinement, then (this has to be matched by the residual set of the state of the abstraction and) the labelling is the invisible action τ. Then, some constraints, called *saturation conditions*, are to be satisfied that resemble somehow the three simulation-like conditions of the definition of vertical bisimulation.

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, $T \uplus U$.

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

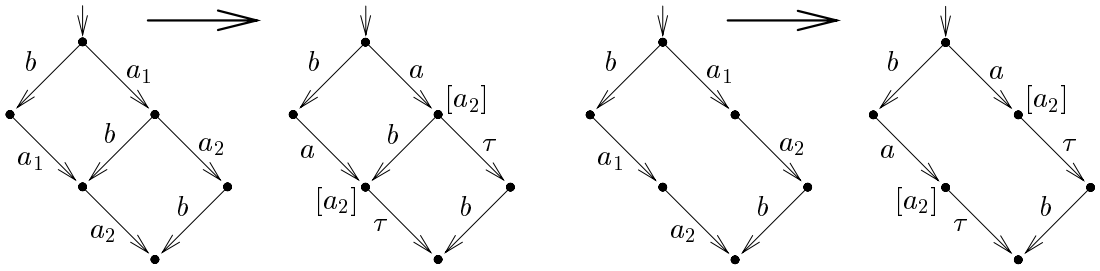
$$\begin{aligned} \to \subseteq\ & \{((s,R),\alpha,(s', R \oplus [v])) \mid s \xrightarrow{\gamma} s', \alpha \in \mathrm{adom}(r), r(\alpha) \xrightarrow{\gamma} v\} \\ & \cup \{((s,R),\tau,(s',R')) \mid s \xrightarrow{\gamma} s', R \xrightarrow{\gamma} R'\} \\ & \cup \{((s,R),\gamma,(s',R)) \mid s \xrightarrow{\gamma} s', \gamma \notin \mathrm{arng}(r)\}\ . \end{aligned}$$

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 $s \stackrel{\sigma}{\Longrightarrow} s''$ such that $(s,R) \stackrel{\alpha}{\Longrightarrow} (s'',\emptyset) \approx (s',R')$;
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 $\exists s \stackrel{\gamma}{\Longrightarrow} s'$ such that $(s,R) \stackrel{\varepsilon}{\Longrightarrow} (s',R') \approx (s,R)$.

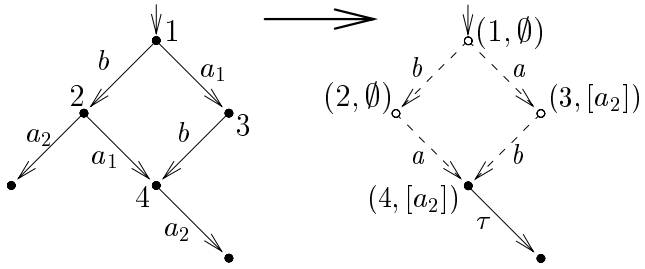
Before showing the formal connection between abstraction and vertical bisimulation, we give a few examples.

5.2 Example. 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).

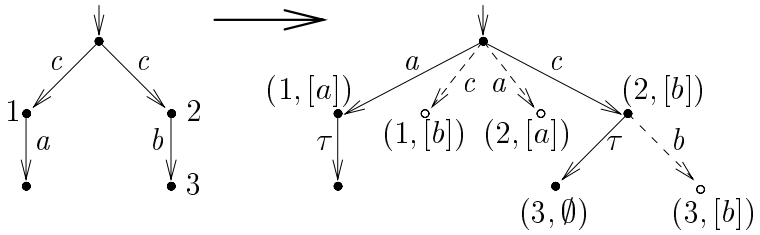


Roughly speaking, the algorithm to follow in order to produce the abstraction of an implementation U, up to some refinement function r, is as follows: first, build all the states and transitions that are reachable from the initial state $(q_U, \emptyset)$; then, check if the saturation conditions are 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.

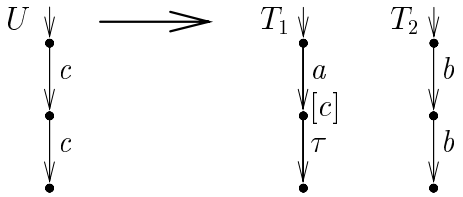
5.3 Example. The following transition system has no abstraction up to $r\colon a \mapsto a_1; a_2$. Consider: if T is an r-abstraction, then the pair $(2, \emptyset)$ violates condition 2 and $(3, [a_2])$ violates condition 3 above, and hence neither is in S_T; therefore, $(1, \emptyset)$ does not satisfy 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.)



5.4 Example. 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 $r\colon a \mapsto c; c,\ b \mapsto c$:

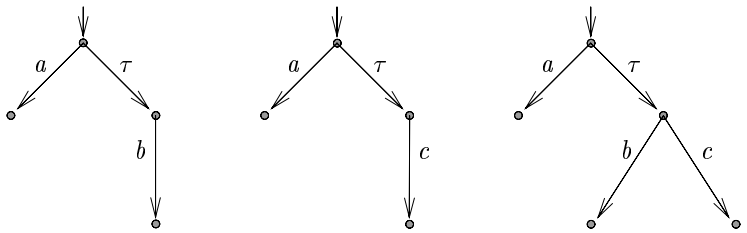


The following theorem states that vertical bisimulation is implied by rooted weak bisimulation w.r.t. some abstraction.

5.5 Theorem. $T \lesssim^r U$ if there exists an r-abstraction V of U such that $T \simeq V$.

The proof can be found in Appendix A (Page 51). 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 5.3 shows, the abstraction of a transition system is not always defined, and as Example 5.4 shows, it may not be unique when it is defined, and may be non-trivial 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 afterwards 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.

5.6 Example. 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 2.9. The reason for this follows from the next lemma.

5.7 Lemma. Let $r\colon \mathbf{A} \to \mathbf{R_C}$ 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 5.5 also holds. The proof can be found in Appendix A (Page 52).

5.8 Theorem. 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 if 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. The next observation is that the abstraction of a finite-state system up to a distinct refinement function is bound to be finite-state.

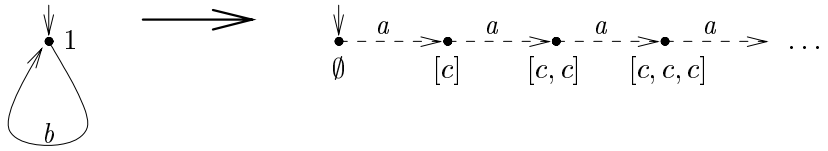
5.9 Lemma. If r is distinct and U is finite-state with r-abstraction T, then T is finite-state.

Proof. Suppose S_T to be infinite; then there is a state $s \in S_U$ for which there is an infinite set $\{(s, R_i)\}_{i\in\mathbb{N}} \subseteq S_T$. For each (s, R_i), let $\sigma_i \in \mathrm{arng}(r)^*$ be the shortest string such that $R_i \stackrel{\sigma_i}{\Longrightarrow} \emptyset$; then $\exists s \stackrel{\sigma_i}{\Longrightarrow} s_i$ such that $(s_i, \emptyset) \in S_T$. Note that $|\sigma_i| \geq |R_i|$ because $v \not\stackrel{\checkmark}{\longrightarrow}$ for all $v \in R_i$. Because there are infinitely many R_i's but only finitely many $\alpha \in \mathrm{adom}(r)$ from which a residual can be derived and only finitely many residuals of any particular $r(\alpha)$, the size of the R_i is unbounded (i.e., for all natural numbers n there is an i such that R_i contains at least n elements); hence it cannot be the case that $|\sigma_i| = |\sigma_j|$ for all $i, j \in \mathbb{N}$. Let i, j be such that $|\sigma_i| < |\sigma_j|$. Due to $s \stackrel{\sigma_i}{\Longrightarrow} s_i$ and saturation condition 2 of Definition 5.1 applied to $(s, R_j) \in S_T$, bearing in mind that (due to Lemma 5.7.3) $r(\alpha) \not\stackrel{c}{\longrightarrow}$ for any action c occurring in σ_i, it follows that there is a transition $R_j \stackrel{\sigma_i}{\Longrightarrow} R'$ such that $(s_i, R') \in S_T$.

Assume $R' \stackrel{\gamma}{\longrightarrow} R''$; then $s_i \stackrel{\gamma}{\longrightarrow} s_i'$ due to saturation condition 3 of Definition 5.1 and $\nexists \alpha \in \mathrm{adom}(r)\colon r(\alpha) \stackrel{\gamma}{\longrightarrow}$ due to Lemma 5.7.3. Since $\gamma \in \mathrm{arng}(r)$, this contradicts condition 2 of Definition 5.1 applied to $(s_i, \emptyset) \in S_T$. Hence we have $R' \not\stackrel{\gamma}{\longrightarrow}$, implying $R' = \emptyset$, which contradicts the construction of σ_j. □

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.

5.10 Example. The following transition system has no abstraction up to $r\colon a \mapsto b; c$, 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.

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

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

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

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

Note that the pre-abstraction of a finite-state transition system need not be finite-state; see Example 5.10. 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 $\exists n \in \mathbb{N}\colon \forall k \geq n\colon U^\omega = U^k$.

The latter observation is shown using an analogous proof to the one of Lemma 5.9. Example 5.10 shows that it does *not* hold if U^ω is not saturated. This brings us to the following algorithm to construct an r-abstraction $U \Uparrow r$ of U.

5.12 Algorithm. Let $r\colon \mathbf{A} \to \mathbf{R_C}$ 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.

5.13 Theorem. Let $r\colon \mathbf{A} \to \mathbf{R_C}$ be a distinct refinement function and U a finite-state transition system with $\Lambda_U = \mathbf{C}_{\tau,\checkmark}$. Algorithm 5.12 succeeds iff $U \Uparrow r$ is an r-abstraction T.

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 following *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 has to extend the relation. The standard open term extension is defined by

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

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:

$$\frac{}{x \sqsubseteq x}$$

Now let us consider the appropriate generalisation to vertical implementation. In order to interpret $t \leq^r u$ where t and u are open terms, we have to 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 have to allow *different* terms to be substituted for x on the left and on the right hand side; 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). But this interpretation still poses problems, as the following example shows.

6.1 Example. Consider the refinement function $r\colon a \mapsto a_1; a_2$. 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 $x\langle x \to t\rangle \leq^r x\langle x \to u\rangle$ (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 $(x/a)\langle x \to t\rangle \leq^{id} (x/a_1, a_2)\langle x \to u\rangle$ (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

$$\frac{}{\mathrm{fv}(t)\colon id \vdash t \sqsubseteq^{id} t}\mathrm{R}_{12} \qquad \frac{X\colon id \vdash t \sqsubseteq^{id} u}{t \sqsubseteq u}\mathrm{R}_{13} \qquad \frac{t \sqsubseteq t' \quad \Gamma \vdash t' \sqsubseteq^r u' \quad u' \sqsubseteq u}{\Gamma \vdash t \sqsubseteq^r u}\mathrm{R}_{14}$$

$$\frac{}{\Gamma \vdash \mathbf{0} \sqsubseteq^r \mathbf{0}}\mathrm{R}_{15} \qquad \frac{}{\Gamma \vdash \mathbf{1} \sqsubseteq^r \mathbf{1}}\mathrm{R}_{16} \qquad \frac{}{\Gamma \vdash \alpha \sqsubseteq^r r(\alpha)}\mathrm{R}_{17} \qquad \frac{\Gamma \vdash t_1 \sqsubseteq^r u_1,\ t_2 \sqsubseteq^r u_2}{\Gamma \vdash t_1 + t_2 \sqsubseteq^r u_1 + u_2}\mathrm{R}_{18}$$

$$\frac{\Gamma \vdash t_1 \sqsubseteq^r u_1,\ t_2 \sqsubseteq^r u_2}{\Gamma \vdash t_1; t_2 \sqsubseteq^r u_1; u_2}\mathrm{R}_{19} \qquad \frac{\Gamma \vdash t \sqsubseteq^r u \quad \phi \restriction \mathrm{adom}(r) = id_{\mathrm{adom}(r)}}{\Gamma \vdash t[\phi] \sqsubseteq^r u[\phi]}\mathrm{R}_{20}$$

$$\frac{\Gamma \vdash t \sqsubseteq^r u \quad r \text{ preserves } A}{\Gamma \vdash t/A \sqsubseteq^{r\setminus A} u/\mathcal{A}(r(A))}\mathrm{R}_{21} \qquad \frac{\Gamma \vdash t_1 \sqsubseteq^r u_1,\ t_2 \sqsubseteq^r u_2 \quad r \text{ is distinct on } A}{\Gamma \vdash t_1 \parallel_A t_2 \sqsubseteq^r u_1 \parallel_{\mathcal{A}(r(A))} u_2}\mathrm{R}_{22}$$

$$\frac{x \notin \mathrm{dom}\,\Gamma}{\Gamma, x\colon r \vdash x \sqsubseteq^r x}\mathrm{R}_{23} \qquad \frac{\Gamma, x\colon r \vdash t \sqsubseteq^r u}{\Gamma \vdash \mu x.\, t \sqsubseteq^r \mu x.\, u}\mathrm{R}_{24}$$

Table 6.5: Vertical proof rules for open and recursive terms

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 $x_1\colon r_1, \ldots, x_n\colon r_n$ where $x_i = x_j$ implies $i = j$. We denote $\mathrm{dom}\,\Gamma = \{x_1, \ldots, x_n\}$ and $\Gamma(x_i) = r_i$. Each pair $(x_i\colon r_i) \in \Gamma$ 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'.

On the other hand, 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 $x\colon r \vdash x/a \sqsubseteq^{id} x/a_1, a_2$. 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 $\mathbf{L}$, presented in Table 3.3 (Page 13), have to be extended with environments as well. The result is given in Table 6.5. The new rules are R_{23} and R_{24}. It is straightforward to show that the following properties hold:

6.2 Proposition.

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 $\Gamma, x\colon r' \vdash t \sqsubseteq^r u$ and $\Gamma \vdash t' \sqsubseteq^{r'} u'$, then $\Gamma \vdash t\langle x \triangleright t' \rangle \sqsubseteq^r u\langle x \triangleright u' \rangle$ (*substitutivity*).

6.3 Example. Let $r\colon a \mapsto a_1; a_2$. The following is a derivation of $\mu x.\, a; x \between\!\between\!\between b \sqsubseteq^r \mu x.\, a_1; a_2; x \between\!\between\!\between b$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x\colon r \vdash a \sqsubseteq^r a_1; a_2}\mathrm{R}_{17} \quad \dfrac{}{x\colon r \vdash x \sqsubseteq^r x}\mathrm{R}_{23}}{x\colon r \vdash a; x \sqsubseteq^r a_1; a_2; x}\mathrm{R}_{19} \quad \dfrac{}{x\colon r \vdash b \sqsubseteq^r b}\mathrm{R}_{17}}{x\colon r \vdash a; x \between\!\between\!\between b \sqsubseteq^r a_1; a_2; x \between\!\between\!\between b}\mathrm{R}_{22}}{\vdash \mu x.\, a; x \between\!\between\!\between b \;\sqsubseteq^r\; \mu x.\, a_1; a_2; x \between\!\between\!\between b}\mathrm{R}_{24}$$

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 as follows:

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

where $f \lesssim^\Gamma g$ abbreviates $\forall x \in \mathrm{dom}(f) = \mathrm{dom}(g) = \mathrm{dom}\,\Gamma\colon f(x) \lesssim^{\Gamma(x)} g(x)$. Comparing (3) with (2), a prominent difference is the fact that in the extension of vertical bisimilarity, the terms on the left hand ("abstract") and the right hand ("concrete") 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_{23} in Table 6.5 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_{24} of Table 6.5, is quite difficult to prove. The usual proof techniques for this kind of congruence property, such as the *up-to* technique proposed in [20, 21] or Howe's technique used in functional calculi [17, 30], are not applicable to non-reflexive, non-transitive relations like vertical bisimilarity. Furthermore, we have investigated alternative extensions of closed relations to open terms in [27]; however, the resulting theory is neither developed far enough nor simple enough to apply here. For that reason, we resort to the sub-language of *strictly well-guarded terms*; on this sub-language, which gives rise to image-finite systems only (w.r.t. the weak transition relation), we can use an inductive characterisation of vertical bisimulation that allows to prove the desired recursion congruence property.

6.4 Definition. First we define under what circumstances we call a variable $x \in \mathbf{X}$ *strictly guarded* in an arbitrary (open) term.

- x is strictly guarded in $\mathbf{0}$ and in a for all $a \in \mathbf{A}$;
- x is strictly guarded in $t + u$ and $t \between_A u$ iff x is strictly guarded in t and in u;
- x is strictly guarded in $t; u$ either if x is strictly guarded in t or if $\mathrm{fv}(t) = \emptyset$ and x is strictly guarded in u;
- x is strictly guarded in $t[\phi]$ and $\mu y.\, t$ iff x is strictly guarded in t;
- x is *not* strictly guarded in $\mathbf{1}$, in y (for all $y \in \mathbf{X}$) or in t/A.

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 $\mu x.\, u$ does not occur within the context of a hiding operator.

The set of strictly well-guarded terms will be denoted $\mathbb{L}^{\mathsf{swg}}_{\mathbf{A}}$ (and accordingly $\mathbf{L}^{\mathsf{swg}}_{\mathbf{A}}$); 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 2.2.

6.5 Proposition. 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 $t\langle y \!\!\rightarrowtail u\rangle$.
2. If t and u are strictly well-guarded, then so is $t\langle y \!\!\rightarrowtail u\rangle$.

The following proposition states the operational consequences of strict guardedness: the strict guardedness of a variable is preserved by internal transitions, and the strict well-guardedness of a term is preserved by any transition (just like ordinary well-guardedness; compare Proposition 2.3).

6.6 Proposition. Let $t, u \in \mathbb{L}$ and $x \in \mathbf{X}$.

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 $s \stackrel{a}{\Longrightarrow} s'$ is finite. The following property formalises the claim, already made above, that strict guardedness is enough to guarantee strict image-finiteness.

6.7 Proposition. 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 characterisation of vertical bisimilarity, applying the principle of stratification seen in [20], except that —due to the strict image-finiteness of the systems we consider— we need only countably many approximations.

6.8 Theorem. 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 $s_1 \xrightarrow{\alpha} s_1'$, then one of the following holds:
    1. $\alpha \in \mathrm{adom}(r)$, and $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$ with $|\sigma| \geq i$ implies $\exists s_2 \stackrel{\sigma}{\Longrightarrow} s_2'$ such that $s_1' \preccurlyeq_{i-|\sigma'|}^{r,\emptyset} s_2'$.
    2. $\alpha \notin \mathrm{adom}(r)$ and $\exists s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2'$ such that $s_1' \preccurlyeq_{i-1}^{r,\emptyset} s_2'$.
  - if $s_2 \xrightarrow{\gamma} s_2'$ then one of the following holds:
    1. $\exists \alpha \in \mathrm{adom}(r).\ \exists s_1 \stackrel{\alpha}{\Longrightarrow} s_1'$ and $\exists r(\alpha) \xrightarrow{\gamma} v$ such that $s_1' \preccurlyeq_{i-1}^{r,R\oplus[v]} s_2'$.
    2. $\exists s_1 \stackrel{\varepsilon}{\Longrightarrow} s_1'$ and $\exists R \xrightarrow{\gamma} R'$ such that $s_1' \preccurlyeq_{i-1}^{r,R'} s_2'$;
    3. $\gamma \notin \mathrm{arng}(r)$ and $\exists s_1 \stackrel{\hat{\gamma}}{\Longrightarrow} s_1'$ such that $s_1' \preccurlyeq_{i-1}^{r,R} s_2'$.
  - If $R \xrightarrow{\gamma} R'$ then $\exists s_2 \stackrel{\gamma}{\Longrightarrow} s_2'$ and $s_1 \preccurlyeq_{i-1}^{r,R'} s_2'$.
- For all $i > 0$, $\lesssim_i^r \subseteq S \times S$ is the largest bi-root of $\preccurlyeq_{i-1}^{r,\emptyset}$.

The proof is postponed to Appendix A.3 (Page 53). We now show that $t \lesssim^r_{x:r} u$ implies $\mu x.\, t \lesssim^{r,\emptyset}_i \mu x.\, u$ for all $i \in \mathbb{N}$; this then implies $\mu x.\, t \lesssim^r \mu x.\, u$. For this purpose, we define the *approximations* of a recursive term $\mu x.\, t$, in the standard way:

$$\begin{aligned} \mu^0 x.\, t &= \mathbf{0} \\ \mu^{i+1} x.\, t &= t\langle x \to \mu^i x.\, t\rangle \ . \end{aligned}$$

By induction on i and the fact that $\mathbf{0} \lesssim^r \mathbf{0}$, it is straightforward to see that $t \lesssim^r_{x:r} u$ implies $\mu^i x.\, t \lesssim^r \mu^i x.\, u$ 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.

6.9 Definition. 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 Appendix A.3). 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.

6.10 Lemma. 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.

6.11 Lemma. 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.

6.12 Theorem. 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 6.11 we have

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

By Lemma 6.10.2, it follows that $\mu x.\, t \lesssim^r_i \mu x.\, u$ for all $i \in \mathbb{N}$. According to Theorem 6.8, 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 6.5 with respect to vertical bisimilarity.

6.13 Theorem. Rooted vertical bisimilarity satisfies all the rules in Table 6.5.

Proof. The soundness proofs of R_{15}–R_{21} are straightforward extensions of the case for closed terms, since essentially nothing happens with the implementation environments.

R_{12} Assume $f, g: \mathrm{fv}(t) \to \mathbf{L}$. It follows from the closed case (Rule R_1) that $f \lesssim^r_{\mathrm{fv}(t):id} g$ iff $f(x) \simeq g(x)$; hence Rule R_{12} 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 2.6).

R_{13} Assume $t \lesssim^{id}_{\mathrm{fv}(t,u):id} u$. 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 $t\langle f\rangle \simeq u\langle f\rangle$. By definition of the open term extension of $\simeq$ (see (2)), it follows that $t \simeq u$.

R_{14} Assume $t \simeq t' \lesssim^r_\Gamma u' \simeq u$. 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_{23} Immediate, by definition of $\lesssim^r$ over open terms; see (3).

R_{24} Proved in Theorem 6.12. □

It is also not difficult to see that the following semantic counterpart to the Proposition 6.2 holds:

6.14 Proposition.

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 $t\langle f\rangle \lesssim^r_{\Gamma'} u\langle g\rangle$.

Proof. By applying the definition of the open term extension $\lesssim^r_\Gamma$, given in (3).

As a final result, note that Corollary 4.12, concerning the correctness of syntactic refinement modulo vertical bisimilarity, can immediately be generalised to the language with recursion. See Table 3.4 (Page 17) for the definition of $r^*: \mathbb{L} \to \mathbb{L}$.

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

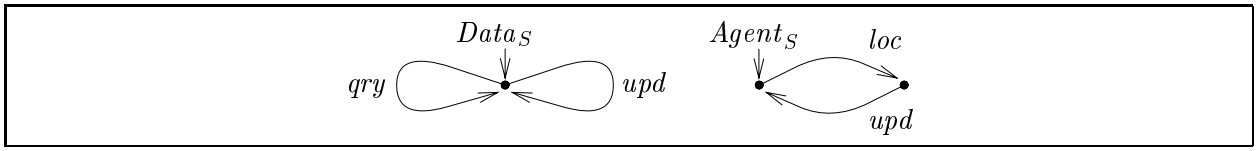


Figure 6: Specification of data base and agent

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, Jonsson and Orava in [6]. Next, we extend the theory with another rule, which expresses that sequential composition on the abstract level may be implemented by a partial overlap on the concrete level. Using this rule, we extend the data base example, and we consider a refinement-driven design step of a booking agent, inspired by Wehrheim [40].

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 modelled by the transition systems depicted in Figure 6.

The problem considered in [6] 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 $r\colon upd \mapsto req; cnf$. 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 behaviour shown in Figure 7.

It is seen that, similar to our approach, the proposed implementations differ from the corresponding specifications in the level of abstraction of their alphabets. The correctness criterion employed in [6] circumvents the associated problems by just requiring (horizontal) correctness *after hiding* the relevant actions: i.e., they prove

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

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

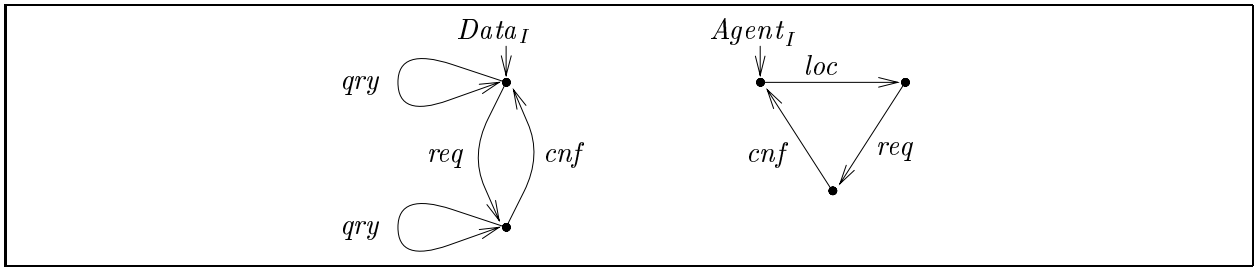


Figure 7: Implementation of data base and agent

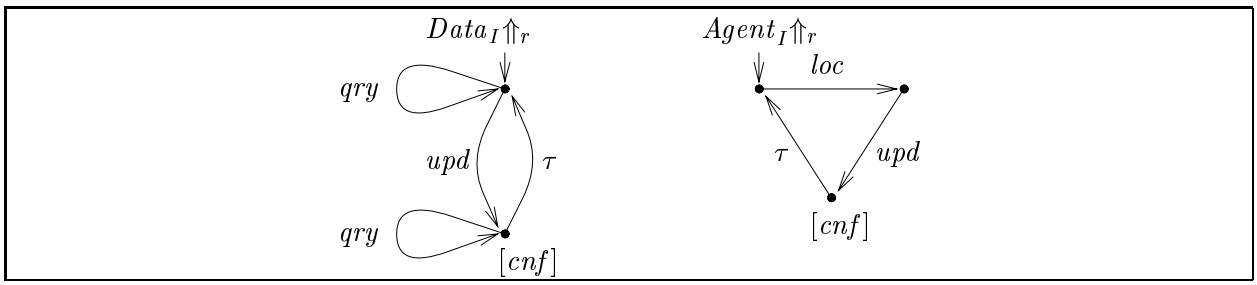


Figure 8: Abstraction of the data base implementation of Figure 7

actions that are changed; for it is not difficult to see that the following hold:

$$\begin{array}{rcl} Data_S & \lesssim^r & Data_I \\ Agent_S & \lesssim^r & Agent_I \end{array}$$

Moreover, we have an effective way to check this, through the Abstraction Theorem 5.5, by constructing $Data_I \Uparrow r$ and $Agent_I \Uparrow r$ (see Figure 8) 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 [6], such a derivation is possible for *Data* but not for *Agent*.) Consider the following algebraic specifications:

$$\begin{array}{rcl} Data_S & = & (\mu x.\ qry; x) \,|||\, (\mu y.\ upd; y) \\ Agent_S & = & \mu z.\ upd; z + loc; z \\ \\ Data_I & = & (\mu x.\ qry; x) \,|||\, (\mu y.\ req; cnf; y) \\ Agent_I & = & \mu z.\ req; cnf; z + loc; z \end{array}$$

The correctness of the *Data*-part can be shown as follows:

$$\dfrac{\dfrac{\vdots}{y{:}\, r \vdash \mu x.\ qry; x \sqsubseteq^r \mu x.\ qry; x} \quad \dfrac{\dfrac{\dfrac{}{y{:}\, r \vdash upd \sqsubseteq^r req; cnf}\mathrm{R}_{17} \quad \dfrac{}{y{:}\, r \vdash y \sqsubseteq^r y}\mathrm{R}_{23}}{y{:}\, r \vdash upd; y \sqsubseteq^r req; cnf; y}\mathrm{R}_{19}}{\vdash \mu y.\ upd; y \sqsubseteq^r \mu y.\ req; cnf; y}\mathrm{R}_{24}}{\vdash Data_S \sqsubseteq^r Data_I}\mathrm{R}_{22}$$

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

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash Data_S \sqsubseteq^r Data_I} \quad \dfrac{\vdots}{\vdash Agent_S \sqsubseteq^r Agent_I}}{\vdash Data_S \parallel_{upd} Agent_S \sqsubseteq^r Data_I \parallel_{req,cnf} Agent_I}\mathrm{R}_{11}}{\vdash (Data_S \parallel_{upd} Agent_S)/upd \sqsubseteq^{id} (Data_I \parallel_{req,cnf} Agent_I)/req, cnf}\mathrm{R}_{10}}{(Data_S \parallel_{upd} Agent_S)/upd \sqsubseteq (Data_I \parallel_{req,cnf} Agent_I)/req, cnf}\mathrm{R}_{2}$$

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.

Using 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: [13] gives conditions under which syntactic substitution coincides with semantic refinement, and it so happens that these conditions are satisfied in the present example. All the same, in comparison to the traditional approach the method proposed in this paper offers the following advantages:

- Our method is based on an interleaving semantics, which means that the results are equally valid when expressed using the transition systems in which the original problem was posed as using the corresponding language description. Not so for traditional action refinement, which is not compatible with interleaving semantics. Instead, there a more "precise" specification has to be given than can be done using transition systems: either a term or 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); under no circumstances will it allow both. In other words, in the traditional approach, the *design decision* of choosing the desired implementation for the data base is taken at an earlier stage, namely as soon as the refinement function is given.

- More importantly, our method allows 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. This means that our notion of vertical implementation can be integrated into existing interleaving-based design methods. There is no similar concept in the traditional approach to action refinement.

It is clear that the above is only a toy example; for instance, the data base has only a single state. Below, we generalise the example to deal with multiple states, but first we extend the algebraic theory with an important implementation principle.

7.2 Sequential composition as partial overlap

Consider the following rule:

$$\frac{r(a) = u_1; u_2 \quad \Gamma \vdash t \sqsubseteq^r v}{\Gamma \vdash a; t \sqsubseteq^r u_1; (u_2 \parallel\!\!\mid v)} \mathrm{R}_{25}$$

This states that under 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, reminiscent of the approach of Janssen, Poel and Zwiers [19] and Wehrheim [40]. It turns out that Rule R_{25} is in fact satisfied by vertical bisimulation.

7.1 Theorem. $\lesssim^r$ satisfies Rule R_{25}.

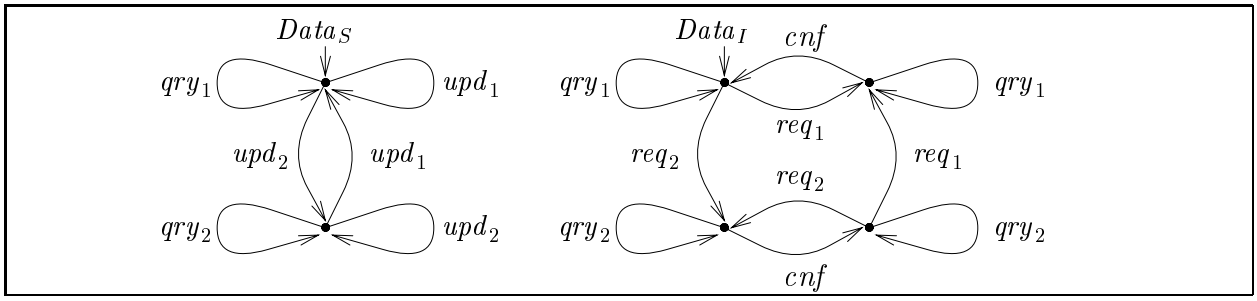


Figure 9: Specification and implementation of a data base with 2 states

For the proof see Appendix A (Page 57). Note that the following, stronger variation on Rule R_{25}, which directly generalises Rule R_{19} from Table 6.5, is *not* satisfied by vertical bisimulation:

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

In this variation, we know nothing about s; it may be the case that the separation into the sequentially composed u_1 and u_2 is inherited from s, i.e., $s = s_1; s_2$ such that $\Gamma \vdash s_i \sqsubseteq^r u_i$ for $i = 1, 2$. Then the latter rule would allow $s_1; s_2; t$ to be implemented by $u_1; (u_2 \,|||\, v)$: in particular, in the special case that $r = id$, this would imply $s_1; s_2; t \sqsubseteq^{id} s_1; (s_2 \,|||\, t)$ and hence (applying R_2) $s_1; s_2; t \sqsubseteq s_1; (s_2 \,|||\, t)$, which, as far as we know, is not satisfied by any implementation relation in the literature.

The use of Rule R_{25} is fairly limited; in the conclusion of this paper we discuss another, more general rule for the weakening of sequential composition. Nevertheless, there are non-trivial applications even of this limited version, as we show in the next example.

7.3 The data base revisited: Multiple states

We now consider a slightly more realistic version of the above data base, where 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 following specification:

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

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 behaviour of $Data_S$ is depicted in Figure 9.

The refinement consists of splitting the update actions as before:

$$r\colon upd_i \mapsto req_i; cnf$$

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

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

The behaviour of $Data_I$ for the case $n = 2$ is also shown in Figure 9. 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:

$$\dfrac{\dfrac{}{y{:}\, r \vdash upd_i \sqsubseteq^r req_i; cnf}\,\mathrm{R}_6 \quad \dfrac{\vdots}{y{:}\, r \vdash Query_i \sqsubseteq^r Query_i}}{y{:}\, r \vdash upd_i \sqsubseteq^r req_i; (cnf \;|||\; Query_i)}\,\mathrm{R}_{25}$$

Note that, as before, there are many possible implementations of $Data_S$; in particular, the completely sequentialised version is also correct:

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

7.4 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:

$$\begin{aligned} Agent_S &= \mu x.\, info; (no; x + yes; x) \\ Cust_S &= \mu y.\, info; (\tau; no; (y + \tau) + yes; trip) \end{aligned}$$

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 synchronisation set is given by $A = \{info, yes, no\}$. Note that we have the following 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:

$$\begin{aligned} yes &\mapsto book; (print + mail) \\ no &\mapsto rel \end{aligned}$$

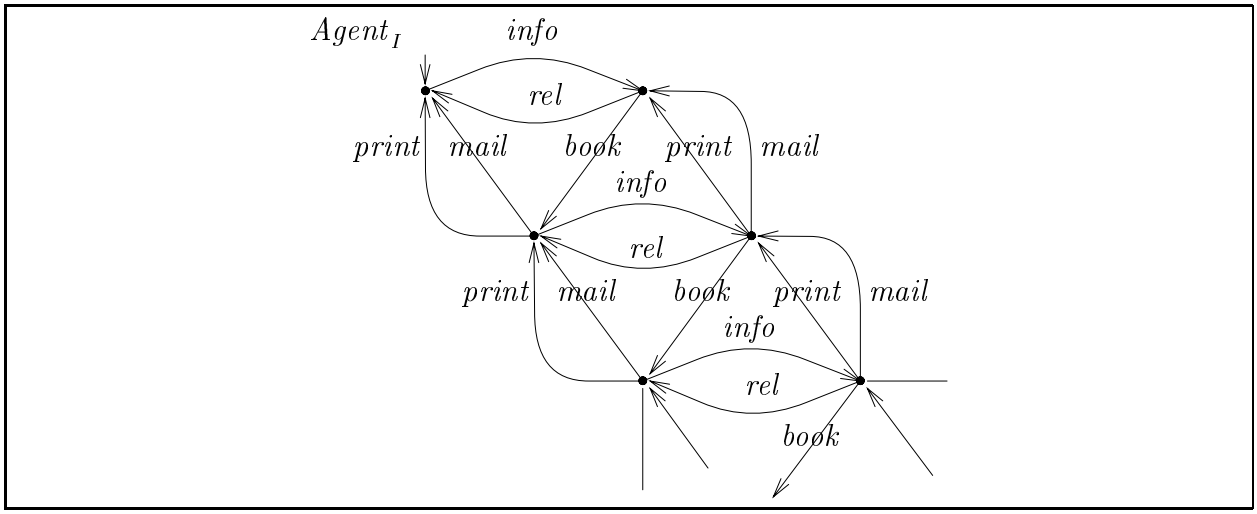


Figure 10: Implementation of the booking agent

The proposed implementation is the following:

$$\begin{aligned} Agent_I &= \mu x.\ info; (rel; x + book; ((print + mail) \,|||\, x)) \\ Cust_I &= \mu x.\ info; (\tau; rel; (y + \tau) + book; (print + mail); trip) \end{aligned}$$

Note that, since there is no bound on the number of outstanding print or mail actions, $Agent_I$ is infinite-state. Figure 10 sketches its behaviour.

Since $Cust_I = r^*(Cust_S)$, it follows by Theorem 6.15 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

$$\dfrac{\dfrac{\vdots}{x{:}\, r \vdash no; x \sqsubseteq^r rel; x}\,\mathrm{R}_{19} \quad \dfrac{\dfrac{}{x{:}\, r \vdash yes \sqsubseteq^r book; (print + mail)}\,\mathrm{R}_{17} \quad \dfrac{}{x{:}\, r \vdash x \sqsubseteq^r x}\,\mathrm{R}_{23}}{x{:}\, r \vdash yes; x \sqsubseteq^r book; ((print + mail) \,|||\, x)}\,\mathrm{R}_{25}}{x{:}\, r \vdash no; x + yes; x \sqsubseteq^r rel; x + book; ((print + mail) \,|||\, x)}\,\mathrm{R}_{18}$$

By R_{22} and R_{21}, 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

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

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 approach to relating 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 work that is even remotely similar is [9], which studies the traditional congruence problem for action refinement with the aim of establishing restrictions under which interleaving models are still compositional; and [14], which considers a different type of action refinement where the refinements are explicitly serialised —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 [13].
- 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 [6]. 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 [14] and later (independently) in [24, 25]. However, the technical material, including the algebraic proof rules and the notion of vertical bisimulation, are completely new.

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.

$$\frac{\Gamma \vdash t \sqsubseteq^{r_1} u,\ u \sqsubseteq^{r_2} v}{\Gamma \vdash t \sqsubseteq^{r_2 \circ r_1} v}\,\mathrm{R}_{26}$$

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 re-introduce the standard congruence problem of the traditional approach to action refinement! Consider:

1. Rule R_6 implies $a \lesssim^{a \mapsto t} t$;
2. Applying Rule R_{26}, 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 4.5 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 substitution, $\lesssim^r$ cannot satisfy Rule R_{26}.

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 $\leq^r$ is based on an interleaving relation $\cong$ and satisfies both Rules R_3 and R_{26}, then either refinement function composition must be defined in some other way, or $a \leq^r t$ may not automatically imply $t \cong r(a)$. In particular, if $\cong$ is rooted bisimilarity as in this paper, a notion of vertical bisimulation satisfying Rule R_{26} must be *weaker* than $\lesssim^r$. On the other hand, it is not difficult to see that weaker versions of $\lesssim^r$ satisfying R_{26} may easily fail to satisfy R_{21} and R_{22}, so that the "solution" would be worse than the problem. For instance, in Section 4, Example 4.4 shows that removing Clause 3 in Definition 4.5 would originate such a problem for R_{21}.

Lax refinement. A problem in the context of action refinement that we have mentioned in the Introduction but rather ignored thereafter 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 $a; b + b; a \lesssim^{a \mapsto a_1; a_2} a_1; a_2 \,|||\, b$). In fact, vertical bisimulation allows a bit more than that, since we have seen that $\lesssim^r$ satisfies Rule R_{25}, 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 its usage 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 the *start* of the implementation of t may overlap with the tail of the refinement of a. This rule unfortunately does not hold for $\lesssim^r$.

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 $r\colon a \mapsto a'; b + a'; c$, 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 $r = r_1 + r_2$).

We intend to develop a notion of "lax vertical bisimulation", satisfying the above rules, in a future paper. We have good hope that the resulting relation will be weak enough to "escape" the problem inherent in vertical composition, discussed above, so that Rule R_{26} can be satisfied as well. Nonetheless, as mentioned in the previous paragraph, the soundness of R_{21} and R_{22} is quite sensitive to weaker versions of $\lesssim^r$; indeed, the current proposal seems to us a reasonable compromise between the different, maybe intrinsically contrasting, desiderata.

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 visualise and straightforward to prove on finite-state systems. However, we feel that any τ-abstracting congruence (see [32] for an overview) would probably also be suitable as a basis for vertical implementation. Natural interesting candidates could be *vertical testing* (see also [24]) and *branching bisimulation* [36]; especially the latter we plan to investigate as it seems to offer a simpler definition; in fact, the reader may have noticed that some of the examples discussed in Section 4.2 compare systems that are (rooted) weak bisimilar but not (rooted) branching bisimilar.

References

[1] L. Aceto and M. C. B. Hennessy. Towards action-refinement in process algebras. *Information and Computation*, 103:204–269, 1993.

[2] L. Aceto and M. C. B. Hennessy. Adding action refinement to a finite process algebra. *Information and Computation*, 115:179–247, 1994.

[3] E. Badouel and P. Darondeau. On guarded recursion. *Theoretical Comput. Sci.*, 82:403–408, 1991.

[4] J. C. M. Baeten and R. J. van Glabbeek. Abstraction and Empty Process in Process Algebra. *Fundamenta Informaticae* XII, pages 221–242, 1989.

[5] J. C. M. Baeten and W. P. Weijland. *Process Algebra.* Cambridge University Press, 1990.

[6] E. Brinksma, B. Jonsson, and F. Orava. Refining interfaces of communicating systems. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT '91, Volume 2*, volume 494 of *Lecture Notes in Computer Science*, pages 297–312. Springer-Verlag, 1991. Also available as Memoranda Informatica 91–19, TIOS 91/003, University of Twente, The Netherlands.

[7] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, July 1984.

[8] L. Castellano, G. De Michelis, and L. Pomello. Concurrency vs. interleaving: An instructive example. *Bull. Eur. Ass. Theoret. Comput. Sci.*, 31:12–15, 1987. Note.

[9] I. Czaja, R. J. van Glabbeek, and U. Goltz. Interleaving semantics and action refinement with atomic choice. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 89–109. Springer-Verlag, 1992. Report version: Arbeitspapiere der GMD 594, Gesellschaft für Mathematik und Datenverarbeitung, No. 1991.

[10] R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Comput. Sci.*, 34:83–133, 1984.

[11] P. Degano and R. Gorrieri. A causal operational semantics of action refinement. *Information and Computation*, 122:97–119, 1995.

[12] P. Degano, R. Gorrieri, and G. Rosolini. A categorical view of process refinement. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*, pages 138–153. Springer-Verlag, 1992.

[13] U. Goltz, R. Gorrieri, and A. Rensink. Comparing syntactic and semantic action refinement. *Information and Computation*, 125(2):118–143, Mar. 1996.

[14] R. Gorrieri. A hierarchy of system descriptions via atomic linear refinement. *Fundamenta Informaticae*, 16:289–336, 1992.

[15] R. Gorrieri and C. Laneve. Split and ST bisimulation semantics. *Information and Computation*, 116(1):272–288, Jan. 1995.

[16] M. C. B. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, Feb. 1995.

[17] D. J. Howe. Proving congruences of bisimulation in functional programming languages. *Information and Computation*, 124:103–112, 1996.

[18] M. Huhn. Action refinement and property inheritance in systems of sequential agents. In U. Montanari and V. Sassone, editors, *Concur '96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 639–654. Springer-Verlag, 1996.

[19] W. Janssen, M. Poel, and J. Zwiers. Actions systems and action refinement in the development of parallel systems. In J. C. M. Baeten and J. F. Groote, editors, *Concur '91*, volume 527 of *Lecture Notes in Computer Science*, pages 298–316. Springer-Verlag, 1991.

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

[21] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[22] M. Nielsen, U. Engberg, and K. G. Larsen. Fully abstract models for a process language with refinement. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 523–549. Springer-Verlag, 1989.

[23] E.-R. Olderog, editor. *Programming Concepts, Methods and Calculi*, volume A–56 of *IFIP Transactions*. IFIP, 1994.

[24] A. Rensink. *Models and Methods for Action Refinement*. PhD thesis, University of Twente, Enschede, Netherlands, Aug. 1993.

[25] A. Rensink. Methodological aspects of action refinement. In Olderog [23], pages 227–246.

[26] A. Rensink. An event-based SOS for a language with refinement. In J. Desel, editor, *Structures in Concurrency Theory*, Workshops in Computing, pages 294–309. Springer-Verlag, 1995.

[27] A. Rensink. Bisimilarity of open terms. In C. Palamidessi and J. Parrow, editors, *Expressiveness in Concurrency*, volume 7 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1997. Full report version: Hildesheimer Informatik-Bericht 5/97, University of Hildesheim, May 1997.

[28] A. Rensink and R. Gorrieri. Action refinement as an implementation relation. In M. Bidoit and M. Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 772–786. Springer-Verlag, 1997.

[29] A. Rensink and H. Wehrheim. Weak sequential composition in process algebras. In B. Jonsson and J. Parrow, editors, *Concur '94: Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 226–241. Springer-Verlag, 1994.

[30] D. Sands. From SOS rules to proof principles: An operational metatheory for functional languages. In *24TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 1997.

[31] R. J. van Glabbeek. The refinement theorem for ST-bisimulation semantics. In *Programming Concepts and Methods*. IFIP, North-Holland Publishing Company, 1990. Report version: RvG–8906, Centre for Mathematics and Computer Science, 1989.

[32] R. J. van Glabbeek. The linear time – branching time spectrum II: The semantics of sequential systems with silent moves. In E. Best, editor, *Concur '93*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1993. Extended abstract. Available as ftp://boole.stanford.edu/spectrum.A4.ps.Z.

[33] R. J. van Glabbeek and U. Goltz. Equivalences and refinement. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990. Report version: SFB-Bericht 423/12/90 A, Technische Universität München, Institut für Informatik.

[34] R. J. van Glabbeek and U. Goltz. Refinement of actions in causality based models. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems — Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 267–300. Springer-Verlag, 1990. Report version: Abeitspapiere der GMD 428.

[35] R. J. van Glabbeek and F. W. Vaandrager. Petri Net models for algebraic theories of concurrency. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE — Parallel Architectures and Languages Europe, Volume II: Parallel Languages*, volume 259 of *Lecture Notes in Computer Science*, pages 224–242. Springer-Verlag, 1987.

[36] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, May 1996.

[37] W. Vogler. Failures semantics based on interval semiwords is a congruence for refinement. *Distributed Computing*, 4:139–162, 1991.

[38] W. Vogler. Bisimulation and action refinement. *Theoretical Comput. Sci.*, 114:173–200, 1993.

[39] W. Vogler. The limit of Split_n-language equivalence. *Information and Computation*, 127(1):41–61, 1996. Report version: No. 288 of the Institut für Mathematik, Universität Augsburg.

[40] H. Wehrheim. Parametric action refinement. In Olderog [23], pages 247–266. Full report version: Hildesheimer Informatik-Berichte 18/93, Institut für Informatik, University of Hildesheim, Nov. 1993.

A Proofs of the theorems

A.1 Proofs of Section 4

We prove the soundness of the rules in Table 3.3 for vertical bisimilarity of closed terms.

4.11 Theorem. $\lesssim^r$ satisfies all the rules in Table 3.3.

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 bi-root 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 bi-root of $\rho^\emptyset$ is a sub-relation 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 bi-root of $\rho^\emptyset$ is a sub-relation 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

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

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 $t_3 \stackrel{\varepsilon}{\Longrightarrow} u_3$ with $u_2 \lesssim^{r,\emptyset} u_3$. We recognise two cases.

1. $\alpha \in \mathrm{adom}(r)$. For all $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$, it follows that $u_3 \stackrel{\sigma}{\Longrightarrow} u_3'$ with $u_2' \preccurlyeq^{r,\emptyset} u_3'$; hence $u_3' \stackrel{\varepsilon}{\Longrightarrow} t_3'$ with $t_2' \preccurlyeq^{r,\emptyset} t_3'$. Due to $t_3 \stackrel{\sigma}{\Longrightarrow} t_3'$, also $t_4 \stackrel{\sigma}{\Longrightarrow} t_4'$ 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 $u_3 \stackrel{\alpha}{\Longrightarrow} u_3'$ such that $u_2' \preccurlyeq^{r,\emptyset} u_3'$. In either case, it then follows that $u_3' \stackrel{\varepsilon}{\Longrightarrow} t_3'$ such that $t_2' \preccurlyeq^{r,\emptyset} t_3'$. Due to $t_3 \stackrel{\hat{\alpha}}{\Longrightarrow} t_3'$, also $t_4 \stackrel{\hat{\alpha}}{\Longrightarrow} t_4'$ 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

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

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

1. $\exists \alpha \in \mathrm{adom}(r)\colon u_2 \stackrel{\alpha}{\Longrightarrow} u_2'$ and $r(\alpha) \xrightarrow{\gamma} v$ with $u_2' \lesssim^{r,R\oplus[v]} u_3'$. It then follows that $u_2' \stackrel{\varepsilon}{\Longrightarrow} t_2'$ such that $t_2' \preccurlyeq^{r,R\oplus[v]} t_3'$. Due to $t_2 \stackrel{\alpha}{\Longrightarrow} t_2'$, also $t_1 \stackrel{\alpha}{\Longrightarrow} t_1'$ such that $t_1' \approx t_2'$. It follows that $t_1'\ \rho^{R\oplus[v]}\ t_4'$.
2. $u_2 \stackrel{\varepsilon}{\Longrightarrow} u_2'$ and $R \xrightarrow{\gamma} R'$ with $u_2' \lesssim^{r,R'} u_3'$. It then follows that $u_2' \stackrel{\varepsilon}{\Longrightarrow} t_2'$ such that $t_2' \preccurlyeq^{r,R'} t_3'$. Due to $t_2 \stackrel{\varepsilon}{\Longrightarrow} t_2'$, also $t_1 \stackrel{\varepsilon}{\Longrightarrow} t_1'$ 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 $u_2 \stackrel{\gamma}{\Longrightarrow} u_2'$ such that $u_2' \preccurlyeq^{r,R} u_3'$. In either case, it then follows that $u_2' \stackrel{\varepsilon}{\Longrightarrow} t_2'$ such that $t_2' \preccurlyeq^{r,R} t_3'$. Due to $t_2 \stackrel{\hat{\gamma}}{\Longrightarrow} t_2'$, also $t_1 \stackrel{\hat{\gamma}}{\Longrightarrow} t_1'$ 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 bi-root 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 bi-root of $\rho^\emptyset$.

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

$$\{(\alpha, r(\alpha), \emptyset), (\mathbf{0}, \mathbf{0}, \emptyset)\} \cup \{(\mathbf{1}, t, [t]) \mid \exists \sigma \in \mathbf{U}^+ \colon r(\alpha) \stackrel{\sigma}{\Longrightarrow} t\}$$

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

R_9 Assume $\phi \restriction \mathrm{adom}(r) = id_{\mathrm{adom}(r)}$. Then 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 a weak vertical bisimulation relation up to r, and $\{(t[\phi], u[\phi]) \mid t \lesssim^r u\}$ is a bi-root of $\rho^\emptyset$.

Injectivity of recognise

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

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

We first prove that $\rho = \{(t/A, u/C, R \setminus C) \mid t \preccurlyeq^{r,R} u\}$ 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 $R \setminus C = \emptyset$. Moreover, assume $t/A \xrightarrow{\alpha} t'/A$.

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 $\mathbf{R}$), $R \stackrel{\sigma}{\Longrightarrow} \emptyset$ for some $\sigma \in C^*$. By the fact that $\{(R, u) \mid t \preccurlyeq^{r,R} u\}$ is a weak simulation, it follows that $u \stackrel{\sigma}{\Longrightarrow} u'$ such that $t \preccurlyeq^{r,\emptyset} u'$. We now recognise two cases.

1. $\alpha \in \mathrm{adom}(r)$; hence $\alpha \notin A$ and $t \xrightarrow{\alpha} t'$. For all $(r \backslash A)(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$, also $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$; hence $u' \stackrel{\sigma}{\Longrightarrow} u''$ such that $t' \preccurlyeq^{r,\emptyset} u''$. It follows that $\sigma \in (\mathbf{C} \setminus C)^*$ and hence $u'/C \stackrel{\sigma}{\Longrightarrow} u''/C$ and $t'/A \; \rho^\emptyset \; u''/C$.
2. $\alpha \notin \mathrm{adom}(r)$. There are two sub-cases.
   - $\alpha = \tau$ and $t \xrightarrow{\beta} t'$ for some $\beta \in A$. Let $r(\beta) \stackrel{\sigma\checkmark}{\Longrightarrow}$ (such a transition always exists due to $r(\beta) \in \mathbf{R}$); then $u' \stackrel{\sigma}{\Longrightarrow} u''$ such that $t' \preccurlyeq^{r,\emptyset} u''$. Since $\sigma \in C^*$, it follows that $u'/C \stackrel{\varepsilon}{\Longrightarrow} u''/C$ and $t'/A \; \rho^\emptyset \; u''/C$.
   - $t \xrightarrow{\alpha} t'$. Then $u' \stackrel{\hat{\alpha}}{\Longrightarrow} u''$ such that $t' \preccurlyeq^{r,\emptyset} u''$; since $\alpha \notin A$, it follows that $u'/C \stackrel{\hat{\alpha}}{\Longrightarrow} u''/C$ 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 $R_0 \setminus C = R$. Moreover, assume $u/C \xrightarrow{\gamma} u'/C$. We recognise two cases.

- $u \xrightarrow{\gamma} u'$ and $\gamma \notin C$. There are three sub-cases.
  1. $\exists \alpha \in \mathrm{adom}(r)\colon t \xRightarrow{\alpha} t'$ 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 $(R_0 \oplus [v]) \setminus C = R \oplus [v]$; hence $t/A \xRightarrow{\alpha} t'/A$ and $(r\backslash A)(\alpha) \xrightarrow{\gamma} v$ such that $t'/A\ \rho^{R \oplus [v]}\ u'/C$.
  2. $t \xRightarrow{\varepsilon} t'$ 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, $R' = R_0' \setminus C$. We may conclude $t/A \xRightarrow{\varepsilon} t'/A$ and $t/A \xRightarrow{\varepsilon} t'/A$ and $t'/A\ \rho^{R'}\ u'/C$.
  3. $\gamma \notin \mathrm{arng}(r)$ and $t \xRightarrow{\hat{\gamma}} t'$ such that $t' \preccurlyeq^{r,R_0} u'$. It follows that $\gamma \notin \mathrm{arng}(r\backslash A)$; moreover, $t/A \xRightarrow{\hat{\gamma}} t'/A$ 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 sub-cases.
  1. $\exists \alpha \in \mathrm{adom}(r)\colon t \xRightarrow{\alpha} t'$ 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 $(R_0' \oplus [v]) \setminus C = R$, implying $t/A \xRightarrow{\varepsilon} t'/A$ and $t'/A\ \rho^R\ u'/C$.
  2. $t \xRightarrow{\varepsilon} t'$ 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 $\mathcal{A}(v) \subseteq C$ and $\mathcal{A}(v') \subseteq C$, and hence $R_0' \setminus C = R_0 \setminus C$. We may conclude $t'/A\ \rho^R\ u'/C$.
  3. $\delta \notin \mathrm{arng}(r)$ and $t \xRightarrow{\hat{\delta}} t'$ such that $t' \preccurlyeq^{r,R_0} u'$. It follows that $\delta \in A$; hence $t/A \xRightarrow{\varepsilon} t'/A$ 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 $R = R_0 \setminus C$ where $t \preccurlyeq^{r,R_0} u$. Now assume $R \xrightarrow{\gamma} R'$. 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 $R' = R_0' \setminus C$. This implies $u \xRightarrow{\gamma} u'$ such that $t \preccurlyeq^{r,R_0'} u'$; we may conclude $u/C \xRightarrow{\gamma} u'/C$ such that $R'\ \kappa\ u'/C$.

It is straightforward to show that $\{(t/A, u/C) \mid t \lesssim^r u\}$ is a bi-root 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

$$\begin{aligned} R \restriction C &= \{v \in R \mid \mathcal{A}(v) \subseteq C\} \\ R \setminus C &= \{v \in R \mid \mathcal{A}(v) \cap C = \emptyset\}\ . \end{aligned}$$

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

$$\rho = \{(t_1 \parallel_A t_2, u_1 \parallel_C u_2, (R_1 \setminus C) \oplus R_2) \mid t_1 \preccurlyeq^{r,R_1} u_1, t_2 \preccurlyeq^{r,R_2} u_2, R_1 \restriction C = R_2 \restriction C\}$$

is a vertical bisimulation relation up to r.

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

- $\alpha \notin A$, $t_1 \xrightarrow{\alpha} t_1'$ and $t_2 = t_2'$. There are two sub-cases.

1. $\alpha \in \mathrm{adom}(r)$, and if $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$, then $u_1 \stackrel{\sigma}{\Longrightarrow} u_1'$ such that $t_1' \preccurlyeq^{r,\emptyset} u_1'$. Since r preserves A, we have $\sigma \in (\mathbf{C} \setminus C)^*$ and hence $u_1 \parallel_C u_2 \stackrel{\sigma}{\Longrightarrow} u_1' \parallel_C u_2'$ 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 $u_1 \stackrel{\hat{\alpha}}{\Longrightarrow} u_1'$ such that $t_1' \preccurlyeq^{r,\emptyset} u_1'$. It follows that $u_1 \parallel_C u_2 \stackrel{\hat{\alpha}}{\Longrightarrow} u_1' \parallel_C u_2'$ 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 $t_2 \stackrel{\alpha}{\longrightarrow} t_2'$. Symmetrical to the case above.

– $\alpha \in A$ and $t_i \stackrel{\alpha}{\longrightarrow} t_i'$ for $i = 1, 2$. Again, there are two sub-cases.

1. $\alpha \in \mathrm{adom}(r)$, and if $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$ then $u_i \stackrel{\sigma}{\Longrightarrow} u_i'$ for $i = 1, 2$ such that $t_i' \preccurlyeq^{r,\emptyset} u_i'$. Since r preserves A, we have $\sigma \in C^*$; hence $u_1 \parallel_C u_2 \stackrel{\sigma}{\Longrightarrow} u_1' \parallel_C u_2'$ and $t_1' \parallel_A t_2' \; \rho^{\emptyset} \; u_1' \parallel_C u_2'$.
2. $\alpha \notin \mathrm{adom}(r)$ and $u_i \stackrel{\hat{\alpha}}{\Longrightarrow} u_i'$ for $i = 1, 2$ such that $t_i' \preccurlyeq^{r,\emptyset} u_i'$. It follows that $u_1 \parallel_C u_2 \stackrel{\hat{\alpha}}{\Longrightarrow} u_1' \parallel_C u_2'$ 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 $R = (R_1 \setminus C) \oplus R_2$. Now assume $u_1 \parallel_A u_2 \stackrel{\gamma}{\longrightarrow} u_1' \parallel_A u_2'$. We recognise three cases.

– $\gamma \notin C$, $u_1 \stackrel{\gamma}{\longrightarrow} u_1'$ and $u_2 = u_2'$. We recognise three further cases.

1. $\exists \alpha \in \mathrm{adom}(r)\colon t_1 \stackrel{\alpha}{\Longrightarrow} t_1'$ and $r(\alpha) \stackrel{\gamma}{\longrightarrow} v$ 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 $((R_1 \oplus [v]) \setminus C) \oplus R_2 = R \oplus [v]$. We may conclude that $t_1 \parallel_A t_2 \stackrel{\alpha}{\Longrightarrow} t_1' \parallel_A t_2'$ with $t_2 = t_2'$ and $t_1' \parallel_A t_2' \; \rho^{R \oplus [v]} \; u_1' \parallel_C u_2'$.
2. $t_1 \stackrel{\varepsilon}{\Longrightarrow} t_1'$ and $R_1 \stackrel{\gamma}{\longrightarrow} R_1'$ 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 $R' = (R_1' \setminus C) \oplus R_2$. It follows that $R \stackrel{\gamma}{\longrightarrow} R'$ and $t_1 \parallel_A t_2 \stackrel{\varepsilon}{\Longrightarrow} t_1' \parallel_A t_2'$ 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 $t_1 \stackrel{\hat{\gamma}}{\Longrightarrow} t_1'$ such that $t_1' \preccurlyeq^{r,R_1} u_1'$. It follows that $t_1 \parallel_A t_2 \stackrel{\hat{\gamma}}{\Longrightarrow} t_1' \parallel_A t_2'$ 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 $u_2 \stackrel{\gamma}{\longrightarrow} u_2'$. Symmetrical to the above case (note that $R = R_1 \oplus (R_2 \setminus C)$.

– $\gamma \in C$ and $u_i \stackrel{\gamma}{\longrightarrow} u_i'$ for $i = 1, 2$. We recognise three further cases.

1. $\exists \alpha \in \mathrm{adom}(r)\colon t_1 \stackrel{\alpha}{\Longrightarrow} t_1'$ and $r(\alpha) \stackrel{\gamma}{\longrightarrow} v$ such that $t_1' \preccurlyeq^{r,R_1 \oplus [v]} u_1'$. Since r is distinct on A, it follows that $R_2 \not\stackrel{\gamma}{\longrightarrow}$ and $r(\alpha') \stackrel{\gamma}{\longrightarrow} v'$ implies $\alpha = \alpha'$ and $v = v'$; hence also $t_2 \stackrel{\alpha}{\Longrightarrow} t_2'$ 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 $((R_1 \oplus [v]) \setminus C) \oplus R_2 \oplus [v] = R \oplus [v]$. We may conclude that $t_1 \parallel_A t_2 \stackrel{\alpha}{\Longrightarrow} t_1' \parallel_A t_2'$ and $t_1' \parallel_A t_2' \; \rho^{R \oplus [v]} \; u_1' \parallel_C u_2'$.
2. $t_1 \stackrel{\varepsilon}{\Longrightarrow} t_1'$ and $R_1 \stackrel{\gamma}{\longrightarrow} R_1'$ such that $t_1' \preccurlyeq^{r,R_1'} u_1'$. Since r is distinct on A, it follows that $r(\alpha) \not\stackrel{\gamma}{\longrightarrow}$ for all $\alpha \in \mathbf{A}$; hence also $t_2 \stackrel{\varepsilon}{\Longrightarrow} t_2'$ and $R_1 \stackrel{\gamma}{\longrightarrow} R_2'$ 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 $v_i \stackrel{\gamma}{\longrightarrow} v_i'$. Again since r is distinct on A, and $R_i \in rsd(r)$ (for $i = 1, 2$) implies $r(\alpha_i) \stackrel{\sigma_i}{\Longrightarrow} v_i$ 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 $R \stackrel{\gamma}{\longrightarrow} R' = R \ominus [v_1] \oplus [v_1'] = (R_1' \setminus C) \oplus R_2'$. We may conclude that $t_1 \parallel_A t_2 \stackrel{\varepsilon}{\Longrightarrow} t_1' \parallel_A t_2'$ and $t_1' \parallel_A t_2' \; \rho^{R \oplus [v]} \; u_1' \parallel_C u_2'$.
3. $\gamma \notin \mathrm{arng}(r)$ and $t_1 \stackrel{\hat{\gamma}}{\Longrightarrow} t_1'$ such that $t_1' \preccurlyeq^{r,R_1} u_1'$. It follows that also $t_2 \stackrel{\hat{\gamma}}{\Longrightarrow} t_2'$ such that $t_2' \preccurlyeq^{r,R_2} u_2'$. We may conclude $t_1 \parallel_A t_2 \stackrel{\hat{\gamma}}{\Longrightarrow} t_1' \parallel_A t_2'$ 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 $R\ \kappa\ u$; then $u = u_1 \parallel_C u_2$ and $R = (R_1 \setminus C) \oplus R_2$ with $R_1 \restriction C = R_2 \restriction C$ and $t_i \preccurlyeq^{r,R_i} u_i$ for $i = 1, 2$. Moreover, assume $R \xrightarrow{\gamma} R'$. It follows that $R' = R \ominus [v] \oplus [v']$ such that $v \xrightarrow{\gamma} v'$. We recognise 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 $R_i \xrightarrow{\gamma} R_i' = R_i \ominus [v] \oplus [v']$ for $i = 1, 2$, implying $u_i \stackrel{\gamma}{\Longrightarrow} u_i'$ such that $t_i \preccurlyeq^{R_i'} u_i'$. Moreover, $R_1' \restriction C = R_2' \restriction C$ and $R' = (R_1' \setminus C) \oplus R_2'$; hence $u_1 \parallel_C u_2 \stackrel{\gamma}{\Longrightarrow} u_1' \parallel_C u_2'$ and $R'\ \kappa\ u_1' \parallel_C u_2'$.
- $\gamma \notin C$; hence (since r preserves A) $\mathcal{A}(v) \cap C = \emptyset$. Assume $v \in R_1$; the case $v \in R_2$ is symmetrical. It follows that $R_1 \xrightarrow{\gamma} R_1' = R_1 \ominus [v] \oplus [v']$, implying $u_1 \stackrel{\gamma}{\Longrightarrow} u_1'$ such that $t_1 \preccurlyeq^{r,R_1'} u_1'$. Moreover, $R_1' \restriction C = R_1 \restriction C = R_2 \restriction C$ and $R' = (R_1' \setminus C) \oplus R_2$; hence $u_1 \parallel_C u_2 \stackrel{\gamma}{\Longrightarrow} u_1' \parallel_C u_2'$ 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 bi-root of $\rho^\emptyset$.

A.2 Proofs of Section 5

5.5 Theorem. $T \lesssim^r U$ if there exists an r-abstraction V of U such that $T \simeq V$.

Proof. Assume that V is an r-abstraction of U (hence the states of V are of the form (s, R) where $s \in S_U$ and R is an r-residual list), and consider $\approx$ over $T \uplus V$. Indications "by sat. i" refer to saturation property i in Definition 5.1. We show that $\rho = \{(s_T, s_U, R) \mid s_T \approx (s_U, R)\}$ is a weak vertical bisimulation relation up to r, proving $T \lesssim^r U$.

Down-simulation. Assume $s_T\ \rho^\emptyset\ s_U$ and $s_T \xrightarrow{\alpha} s_T'$; it follows that $s_T \approx (s_U, \emptyset)$. We recognise two cases:

1. $\alpha \in \mathrm{adom}(r)$. Assume $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$ for $\sigma \in \mathbf{C}^*$. It follows (due to $s_T \approx (s_U, \emptyset)$) that $(s_U, \emptyset) \stackrel{\alpha}{\Longrightarrow} (s_U', R')$ such that $s_T' \approx (s_U', R')$. We can subdivide $(s_U, \emptyset) \stackrel{\alpha}{\Longrightarrow} (s_U', R')$ into
$$(s_U, \emptyset) \stackrel{\varepsilon}{\Longrightarrow} (s_0, R_0) \xrightarrow{\alpha} (s_1, R_1) \stackrel{\varepsilon}{\Longrightarrow} (s_U', R')$$
which implies $s_U \stackrel{\varepsilon}{\Longrightarrow} s_0$ and $R_0 = \emptyset$. By sat. 1 it follows that $\exists s_0 \stackrel{\sigma}{\Longrightarrow} s_1'$ such that $(s_1', \emptyset) \approx (s_1, R_1)$. Then from $(s_1, R_1) \stackrel{\varepsilon}{\Longrightarrow} (s_U', R')$ it follows that $\exists (s_1', \emptyset) \stackrel{\varepsilon}{\Longrightarrow} (s_U'', R'')$ such that $(s_U'', R'') \approx (s_U', R')$; hence $s_1' \stackrel{\varepsilon}{\Longrightarrow} s_U''$ and $R'' = \emptyset$.
It follows that $s_U \stackrel{\sigma}{\Longrightarrow} s_U''$ and $s_T' \approx (s_U'', R'')$; hence $s_T'\ \rho^\emptyset\ s_U''$.

2. $\alpha \notin \mathrm{adom}(r)$. Due to $s_T \approx (s_U, \emptyset)$ we have $\exists (s_U, \emptyset) \stackrel{\hat{\alpha}}{\Longrightarrow} (s_U', R')$ such that $s_T' \approx (s_U', R')$; due to the definition of V, this implies $s_U \stackrel{\hat{\alpha}}{\Longrightarrow} s_U'$ and $R' = \emptyset$; hence $s_T'\ \rho^\emptyset\ s_U'$.

Up-simulation. Assume $s_T\ \rho^R\ s_U$ and $s_U \xrightarrow{\gamma} s_U'$; it follows that $s_T \approx (s_U, R)$. By sat. 2, we recognise three cases:

1. $\exists \alpha \in \mathrm{adom}(r) : r(\alpha) \xrightarrow{\gamma} v$ such that $(s_U, R) \xrightarrow{\alpha} (s_U', R \oplus [v])$. Due to $s_T \approx (s_U, R)$, it follows that $s_T \stackrel{\alpha}{\Longrightarrow} s_T'$ such that $s_T' \approx (s_U', R \oplus [v])$; hence $s_T'\ \rho^{R \oplus [v]}\ s_U'$.

2. $R \xrightarrow{\gamma} R'$ such that $(s_U, R) \xrightarrow{\tau} (s_U', R')$. Again due to $s_T \approx (s_U, R)$, it follows that $s_T \stackrel{\varepsilon}{\Longrightarrow} s_T'$ such that $s_T' \approx (s_U', R')$; hence $s_T'\ \rho^{R'}\ s_U'$.

3. $\gamma \notin \mathrm{arng}(r)$ and $(s_U, R) \xrightarrow{\gamma} (s_U', R)$; hence $s_T \approx (s_U, R)$ implies $s_T \stackrel{\hat{\gamma}}{\Longrightarrow} s_T'$ such that $s_T' \approx (s_U', R)$, implying $s_T'\ \rho^R\ s_U'$.

Residual simulation. We prove that $\kappa = \{(R, s_U) \mid s_T \; \rho^R \; s_U\}$ is a weak simulation for every s_T. Assume $R \; \kappa \; s_U$ and $R \xrightarrow{\gamma} R'$; it follows that $s_T \approx (R, s_U)$. By sat. 3, $\exists s_U \stackrel{\gamma}{\Longrightarrow} s'_U$ such that $(s_U, R) \approx (s'_U, R')$; hence $s_T \; \rho^{R'} \; s'_U$, implying $R' \; \kappa \; s'_U$.

Rooted bisimilarity. The fact that $\{(q_T, q_U)\}$ is a bi-root of $\rho^\emptyset$ immediately follows from the fact that $q_T \simeq (q_U, \emptyset)$, i.e., $(q_T, (q_U, \emptyset))$ is in the largest bi-root of $\approx$ (taking into account that τ-transitions do not affect the residual set). □

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

Proof. We define $V = \langle \mathbf{A}_\tau, S, \rightarrow, (q_U, \emptyset) \rangle$ where $S = \{(s_U, R) \mid \exists s_T \colon s_T \preccurlyeq^{r,R} s_U\}$ and

$$\begin{aligned} \rightarrow = \; & \{((s_U, R), \alpha, (s'_U, R \oplus [v])) \mid s_U \xrightarrow{\gamma} s'_U, \alpha \in \mathrm{adom}(r), r(\alpha) \xrightarrow{\gamma} v\} \\ & \cup \{((s_U, R), \tau, (s'_U, R')) \mid s_U \xrightarrow{\gamma} s'_U, R \xrightarrow{\gamma} R'\} \\ & \cup \{((s_U, \emptyset), \gamma, (s'_U, \emptyset)) \mid s_U \xrightarrow{\gamma} s'_U, \gamma \notin \mathrm{arng}(r)\} \end{aligned}$$

We first show that $T \simeq V$, and afterwards that V is indeed an r-abstraction of U.

Weak bisimilarity. Consider the following definition:

$$\rho = \{(s_T, (s_U, R)) \mid s_T \preccurlyeq^{r,R} s_U\} \; .$$

We show that ρ and ρ^{-1} are weak simulation relations. Assume $s_T \; \rho \; (s_U, R)$; it follows that $s_T \preccurlyeq^{r,R} s_U$.

- Assume $s_T \xrightarrow{\alpha} s'_T$. Note that $R \stackrel{\sigma}{\Longrightarrow} \emptyset$ for some $\sigma \in \mathbf{C}^*$ since $R \in rsd(r)$ is finite and all terms in R are finite (by definition of $\mathbf{R}$). Due to residual bisimulation, $s_U \stackrel{\sigma}{\Longrightarrow} s'_U$ such that $s_T \preccurlyeq^{r,\emptyset} s'_U$; hence by construction of V, $(s_U, R) \stackrel{\varepsilon}{\Longrightarrow} (s'_U, \emptyset)$. We recognise two cases.

  1. $\alpha \in \mathrm{adom}(r)$. Let $r(\alpha) \stackrel{\gamma\sigma\checkmark}{\Longrightarrow}$ be arbitrary (such a transition sequence always exists, by definition of $\mathbf{R}$); then by down-simulation, $s'_U \stackrel{\gamma\sigma}{\Longrightarrow} s''_U$ such that $s'_T \preccurlyeq^{r,\emptyset} s''_U$; hence $s'_T \; \rho \; (s''_U, \emptyset)$.
     Let v be such that $r(\alpha) \xrightarrow{\gamma} v \stackrel{\sigma\checkmark}{\Longrightarrow}$; then $s'_U \stackrel{\gamma\sigma}{\Longrightarrow} s''_U$ can be separated into

     $$s'_U \stackrel{\varepsilon}{\Longrightarrow} s' \xrightarrow{\gamma} s'' \stackrel{\sigma}{\Longrightarrow} s''_U$$

     to which there is an abstracted sequence

     $$(s'_U, \emptyset) \stackrel{\varepsilon}{\Longrightarrow} (s', \emptyset) \xrightarrow{\alpha} (s'', [v]) \stackrel{\varepsilon}{\Longrightarrow} (s''_U, \emptyset) \; .$$

     It follows that $(s_U, R) \stackrel{\alpha}{\Longrightarrow} (s''_U, \emptyset)$.
  2. $\alpha \notin \mathrm{adom}(r)$. Due to down-simulation, $s'_U \stackrel{\hat{\alpha}}{\Longrightarrow} s''_U$ such that $s'_T \; \rho^\emptyset \; s''_U$, implying $s'_T \; \rho \; (s''_U, \emptyset)$; hence (by construction of V) $(s'_U, \emptyset) \stackrel{\hat{\alpha}}{\Longrightarrow} (s''_U, \emptyset)$, implying $(s_U, R) \stackrel{\hat{\alpha}}{\Longrightarrow} (s''_U, \emptyset)$.

- Assume $(s_U, R) \xrightarrow{\alpha} (s'_U, R')$. By the construction of V, there are three possibilities:

  1. $s_U \xrightarrow{\gamma} s'_U$, $\alpha \in \mathrm{adom}(r)$ and $r(\alpha) \xrightarrow{\gamma} v$ with $R' = R \oplus [v]$. Clearly, $\gamma \in \mathrm{arng}(r)$; moreover, by Lemma 5.7.3, it follows that $R \not\xrightarrow{\gamma}$. By up-simulation, therefore, $\exists r(\alpha') \xrightarrow{\gamma} v'$ such that $s_T \stackrel{\alpha'}{\Longrightarrow} s'_T$ and $s'_T \preccurlyeq^{r, R \oplus [v']} s'_U$. Due to Lemma 5.7.1, it follows that $\alpha' = \alpha$ and $v' = v$; hence $s_T \stackrel{\alpha}{\Longrightarrow} s'_T$ and $s'_T \; \rho \; (s'_U, R \oplus [v])$.

2. $s_U \xrightarrow{\gamma} s'_U$, $R \xrightarrow{\gamma} R'$ and $\alpha = \tau$. Clearly, $\gamma \in \mathrm{arng}(r)$; moreover, by Lemma 5.7.3, it follows that $r(\alpha) \not\xrightarrow{\gamma}$ for all $\alpha \in \mathrm{adom}(r)$. By up-simulation, therefore, $s_T \stackrel{\varepsilon}{\Longrightarrow} s'_T$ and $R \xrightarrow{\gamma} R''$ such that $s'_T \preccurlyeq^{r,R''} s'_U$, Due to Lemma 5.7.1, it follows that $R'' = R'$; hence $s_T \stackrel{\hat{\alpha}}{\Longrightarrow} s'_T$ and $s'_T \; \rho \; (s'_U, R')$.

3. $s_U \xrightarrow{\alpha} s'_U$, $\alpha \notin \mathrm{arng}(r)$ and $R' = R$. By up-simulation, $\exists s_T \stackrel{\hat{\alpha}}{\Longrightarrow} s'_T$ such that $s'_T \preccurlyeq^{r,R} s'_U$, implying $s'_T \; \rho \; (s'_U, R)$.

Since $q_T \lesssim^r q_U$, implying (q_T, q_U) is in the largest bi-root of $\preccurlyeq^{r,\emptyset}$, it follows that $\{(q_T, q_U)\}$ is a bi-root of ρ.

Abstraction. We now show that V is an r-abstraction of U; that is, we show that the saturation conditions of Definition 5.1 hold.

1. Assume $(s_U, \emptyset) \xrightarrow{\alpha} (s'_U, R')$ with $\alpha \in \mathrm{adom}(r)$. By construction of V, $s_U \xrightarrow{\gamma} s'_U$ and $r(\alpha) \xrightarrow{\gamma} v$ such that $R' = [v]$. Now assume $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$.

   Clearly, $\gamma \in \mathrm{arng}(r)$; moreover, $\emptyset \not\xrightarrow{\gamma}$. By up-simulation, $s_T \preccurlyeq^{r,\emptyset} s_U$ then implies $\exists r(\alpha') \xrightarrow{\gamma} v'$ and $s_T \stackrel{\alpha'}{\Longrightarrow} s'_T$ for some $\alpha' \in \mathrm{adom}(r)$ such that $s'_T \preccurlyeq^{r,[v']} s'_U$. By Lemma 5.7.1, $\alpha' = \alpha$ and $v' = v$; hence (as proved above) $s'_T \approx (s'_U, R')$.

   The transition $s_T \stackrel{\alpha}{\Longrightarrow} s'_T$ can be subdivided into

   $$s_T \stackrel{\varepsilon}{\Longrightarrow} s'_1 \xrightarrow{\alpha} s''_1 \stackrel{\varepsilon}{\Longrightarrow} s'_T \ .$$

   By down-simulation, there are corresponding transitions

   $$s_U \stackrel{\varepsilon}{\Longrightarrow} s'_2 \stackrel{\sigma}{\Longrightarrow} s''_2 \stackrel{\varepsilon}{\Longrightarrow} s''_U$$

   such that $s'_1 \preccurlyeq^{r,\emptyset} s'_2$, $s''_1 \preccurlyeq^{r,\emptyset} s''_2$ and $s'_T \preccurlyeq^{r,\emptyset} s''_U$. It follows that $s_U \stackrel{\sigma}{\Longrightarrow} s''_U$; moreover (as proved above) $s'_T \approx (s''_U, \emptyset)$ and hence $(s''_U, \emptyset) \approx (s'_U, R')$.

2. Assume $(s_U, R) \in S$ and $s_U \xrightarrow{\gamma} s'_U$; let s_T be such that $s'_T \preccurlyeq^{r,R} s'_U$. By up-simulation, there are three possibilities:

   (a) $s_T \stackrel{\alpha}{\Longrightarrow} s'_T$ and $r(\alpha) \xrightarrow{\gamma} v$ for some $\alpha \in \mathrm{adom}(r)$ such that $s'_T \preccurlyeq^{r,R \oplus [v]} s'_U$. It follows that $(s_U, R) \xrightarrow{\alpha} (s'_U, R \oplus [v])$.

   (b) $s_T \stackrel{\varepsilon}{\Longrightarrow} s'_T$ and $R \xrightarrow{\gamma} R'$ such that $s'_T \preccurlyeq^{r,R'} s'_U$. It follows that $(s_U, R) \xrightarrow{\tau} (s'_U, R')$.

   (c) $\gamma \notin \mathrm{arng}(r)$ and $s_T \stackrel{\hat{\alpha}}{\Longrightarrow} s'_T$ such that $s'_T \preccurlyeq^{r,R} s'_U$ It follows that $(s_U, R) \xrightarrow{\gamma} (s'_U, R)$.

3. Assume $(s_U, R) \in S$ and $R \xrightarrow{\gamma} R'$; let s_T be such that $s_T \preccurlyeq^{r,R} s_U$. By residual simulation, $\exists s_U \stackrel{\gamma}{\Longrightarrow} s'_U$ such that $s_T \preccurlyeq^{r,R'} s'_U$; hence $(s_U, R) \stackrel{\varepsilon}{\Longrightarrow} (s'_U, R')$. It follows by the proof above that $(s_U, R) \approx s_T \approx (s'_U, R')$; hence $(s_U, R) \approx (s'_U, R')$. □

A.3 Proofs of Section 6

We first prove the vertical stratification theorem.

6.8 Theorem. If T is a strict image-finite transition system, then $\preccurlyeq^r = \bigcap_{i \in \mathbb{N}} \preccurlyeq^r_i$ and $\lesssim^r = \bigcap_{i \in \mathbb{N}} \lesssim^r_i$, where $(\preccurlyeq^r_i)_{i \in \mathbb{N}}$ and $(\lesssim^r_i)_{i \in \mathbb{N}}$ 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 $s_1 \xrightarrow{\alpha} s_1'$, then one of the following holds:
    1. $\alpha \in \mathrm{adom}(r)$, and $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$ with $|\sigma| \geq i$ implies $\exists s_2 \stackrel{\sigma}{\Longrightarrow} s_2'$ such that $s_1' \preccurlyeq_{i-|\sigma'|}^{r,\emptyset} s_2'$.
    2. $\alpha \notin \mathrm{adom}(r)$ and $\exists s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2'$ such that $s_1' \preccurlyeq_{i-1}^{r,\emptyset} s_2'$.
  - if $s_2 \xrightarrow{\gamma} s_2'$ then one of the following holds:
    1. $\exists \alpha \in \mathrm{adom}(r).\ \exists s_1 \stackrel{\alpha}{\Longrightarrow} s_1'$ and $\exists r(\alpha) \xrightarrow{\gamma} v$ such that $s_1' \preccurlyeq_{i-1}^{r,R \oplus [v]} s_2'$.
    2. $\exists s_1 \stackrel{\varepsilon}{\Longrightarrow} s_1'$ and $\exists R \xrightarrow{\gamma} R'$ such that $s_1' \preccurlyeq_{i-1}^{r,R'} s_2'$;
    3. $\gamma \notin \mathrm{arng}(r)$ and $\exists s_1 \stackrel{\hat{\gamma}}{\Longrightarrow} s_1'$ such that $s_1' \preccurlyeq_{i-1}^{r,R} s_2'$.
  - If $R \xrightarrow{\gamma} R'$ then $\exists s_2 \stackrel{\gamma}{\Longrightarrow} s_2'$ and $s_1 \preccurlyeq_{i-1}^{r,R'} s_2'$.
- For all $i > 0$, $\lesssim_i^r$ is the largest bi-root of $\preccurlyeq_{i-1}^{r,\emptyset}$.

Proof. First we show $\preccurlyeq^r = \bigcap_{i \in \mathbb{N}} \preccurlyeq_i^r$.

$\subseteq$ By comparing the conditions in the theorem to Definition 4.5, it is clear that for every weak vertical bisimulation relation ρ over T, $\rho \subseteq \preccurlyeq_i^r$ for all $i \in \mathbb{N}$.

$\supseteq$ For this direction, we need the strict image-finiteness of T. We show that the relation $\rho = \bigcap_{i \in \mathbb{N}} \preccurlyeq_i^r$ is a weak vertical bisimulation relation.

1. First, we prove that $\rho^\emptyset$ is a down-simulation. Assume $s_1\ \rho^\emptyset\ s_2$ (hence $s_1 \preccurlyeq_i^{r,\emptyset} s_2$ for all $i \in \mathbb{N}$) and $s_1 \xrightarrow{\alpha} s_1'$. There are two possibilities.
   (a) $\alpha \in \mathrm{adom}(r)$. Then for all $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow}$ and for all $i \geq |\sigma|$, there is a $s_2 \stackrel{\sigma}{\Longrightarrow} s_{2,i}'$ such that $s_1' \preccurlyeq_{i-|\sigma|}^{r,\emptyset} s_{2,i}'$. Due to the strict image-finiteness of T, infinitely many of the $s_{2,i}'$ coincide; that is, there is a s_2' such that
   $$\forall i > 0: \exists j \geq i: s_{2,j}' = s_2' \ .$$
   Since $\preccurlyeq_i^r \supseteq \preccurlyeq_j^r$ if $i \leq j$, it follows that $s_1' \preccurlyeq_{i-|\sigma|}^{r,\emptyset} s_2'$ for all $i \geq |\sigma|$, implying $s_1'\ \rho^\emptyset\ s_2'$.
   (b) $\alpha \notin \mathrm{adom}(r)$. Then for all $i > 0$, there is a $s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_{2,i}'$ such that $s_1' \preccurlyeq_{i-1}^{r,\emptyset} s_{2,i}'$. Similar to the above case, due to the strict image-finiteness of T, it follows that there is an s_2' with which infinitely many of the $s_{2,i}'$ coincide, implying $s_1' \preccurlyeq_{i-1}^{r,\emptyset} s_2'$ for all $i \geq 0$ and hence $s_1'\ \rho^\emptyset\ s_2'$.
2. Then, we prove that ρ is an up-simulation. Assume $s_1\ \rho^R\ s_2$ (hence $s_1 \preccurlyeq_i^{r,R} s_2$ for all $i \in \mathbb{N}$) and $s_2 \xrightarrow{\gamma} s_2'$. It follows that for all $i > 0$, there are three possible cases.
   (a) $\exists \alpha_i \in \mathrm{adom}(r).\ \exists s_1 \stackrel{\alpha_i}{\Longrightarrow} s_{1,i}'$ and $\exists r(\alpha_i) \xrightarrow{\gamma_i} v_i$ such that $s_{1,i}' \preccurlyeq_{i-1}^{r,R \oplus [v_i]} s_2'$.
   (b) $\exists s_1 \stackrel{\varepsilon}{\Longrightarrow} s_{1,i}'$ and $\exists R \xrightarrow{\gamma} R_i'$ such that $s_{1,i}' \preccurlyeq_{i-1}^{r,R_i'} s_2'$;
   (c) $\gamma \notin \mathrm{arng}(r)$ and $\exists s_1 \stackrel{\hat{\gamma}}{\Longrightarrow} s_{1,i}'$ such that $s_{1,i}' \preccurlyeq_{i-1}^{r,R} s_2'$.
   At least one of these cases must occur for infinitely many i. Depending on which case this is, the following arguments apply.

(a) Since $r(\alpha) = \alpha$ for all but a finite number of α, it follows that $r(\alpha) \xrightarrow{\gamma}$ for only a finite number of α. Hence, infinitely many of the α_i coincide; say $\alpha_i = \alpha$ for all $i \in I$, where $I \subseteq \mathbb{N}$ is infinite. Moreover, since $r(\alpha) \in \mathbf{R}$ is finitely branching, infinitely many of the v_i for $i \in I$ coincide; say $v_i = v$ for all $i \in J$, where $J \subseteq I$ is still infinite. Finally, since T is strict image-finite, infinitely many of the $s'_{1,i}$ for $i \in J$ coincide; say $s'_{1,i} = s'_1$ for all $i \in K$, where $K \subseteq J$ is still infinite. Taken together, we have:

$$\forall i \in \mathbb{N}\colon \exists i \leq k \in K\colon v_k = v, s'_{1,k} = s'_1 \ .$$

Since $\preccurlyeq^r_i \supseteq \preccurlyeq^r_j$ if $i \leq j$, it follows that $s'_1 \preccurlyeq^{r,R\oplus[v]}_{i-1} s'_2$ for all $i > 0$, implying $s'_1 \rho^{r,R\oplus[v]} s'_2$.

(b) Again, due to the strict image-finiteness of T, the finiteness of the multiset R and the finite branching of the derivatives of the r-images, infinitely many of the $s'_{1,i}$ and R'_i must coincide, say on s'_1 and R', respectively; similar as in the previous case, this implies $s'_1 \preccurlyeq^{r,R'}_{i-1} s'_2$ for all $i > 0$, and hence $s'_1 \rho^{r,R'} s'_2$.

(c) Analogous to Case 2 of the down-simulation.

3. Finally, we prove that $\kappa = \{(R, s_2) \mid s_1 \ \rho^R \ s_2\}$ is a residual simulation. Assume $R \ \kappa \ s_2$ (hence $s_1 \preccurlyeq^{r,R}_i s_2$ for all $i \in \mathbb{N}$) and $R \xrightarrow{\gamma} R'$. For all $i > 0$, it follows that $s_2 \stackrel{\gamma}{\Longrightarrow} s'_{2,i}$ such that $s_1 \ \rho^{R'} \ s'_{2,i}$. Since T is strict image-finite, infinitely many of the $s'_{2,i}$ must coincide, say on s'_2; similar as in the cases of the down- and up-simulations, this implies $s_1 \preccurlyeq^{r,R'}_i s'_2$ for all $i > 0$, hence $s_1 \ \rho^R \ s'_2$ and $R' \ \kappa \ s'_2$.

Next, we show $\lesssim^r = \bigcap_{i\in\mathbb{N}} \lesssim^r_i$.

$\subseteq$ $\lesssim \subseteq \lesssim^r_0$ is immediate. Furthermore, since $\preccurlyeq \subseteq \preccurlyeq^r_i$ (see above) and $\lesssim^r_{i+1}$ is the largest bi-root of $\preccurlyeq^r_i$ for all $i \in \mathbb{N}$, it also follows that $\lesssim^r \subseteq \lesssim^r_i$ for all $i > 0$.

$\supseteq$ We show that the relation $\tilde{\rho} = \bigcap_{i\in\mathbb{N}} \lesssim^r_i$ is a bi-root of $\preccurlyeq^r$. Assume $s_1 \ \tilde{\rho} \ s_2$ and $s_1 \xrightarrow{\tau} s'_1$. For all $i > 0$, from $s_1 \lesssim^r_i s_2$ it follows that $\exists s_2 \stackrel{\tau}{\Longrightarrow} s'_{2,i}$ such that $s'_1 \preccurlyeq^{r,\emptyset}_{i-1} s'_{2,i}$. Due to the strict image-finiteness of T, there are only finitely many candidates for $s'_{2,i}$; hence infinitely many of them coincide, say $s'_{2,i} = s'_2$ for all $i \in I$ where $I \subseteq \mathbb{N}$ is infinite. It follows that

$$\forall i \in \mathbb{N}\colon \exists i \leq j \in I\colon s'_1 \preccurlyeq^{r,\emptyset}_{i-1} s'_2$$

and since $\lesssim^r_i \supseteq \lesssim^r_j$ if $i \leq j$, it follows that $s'_1 \preccurlyeq^{r,\emptyset}_{i-1} s'_2$ for all $i > 0$, implying $s'_1 \preccurlyeq^r s'_2$ by the first half of this theorem. □

It remains to prove the auxiliary lemmata leading up to Theorem 6.12.

6.10 Lemma. 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 the lemma has been proved for all $j < i$.

1. Assume $s_1 \sim_i s_2 \preccurlyeq^{r,R} s_3 \sim_i s_4$.
   - 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 recognise the following cases.
     (a) $\alpha \in \mathrm{adom}(r)$. Then $r(\alpha) \stackrel{\sigma\checkmark}{\Longrightarrow} v$ implies $\exists s_2 \stackrel{\sigma}{\Longrightarrow} s_3'$ such that $s_2' \preccurlyeq^{r,\emptyset} s_3'$. If $i > |\sigma|$, we may deduce that $s_4 \stackrel{\sigma}{\Longrightarrow} s_4'$ such that $s_3' \sim_{i-|\sigma|} s_4'$. Since $\sim_{i-|\sigma|} \supseteq \sim_{i-1}$, by the induction hypothsis it follows that $s_1' \preccurlyeq^{r,\emptyset}_{i-|\sigma|-1} s_4'$.
     (b) $\alpha \notin \mathrm{arng}(r)$. Then $\exists s_3 \stackrel{\hat{\alpha}}{\Longrightarrow} s_3'$ such that $s_2' \preccurlyeq^{r,\emptyset} s_3'$; since $i > 1$, we may deduce that $s_4 \stackrel{\hat{\alpha}}{\Longrightarrow} s_4'$ 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 $s_4 \xrightarrow{\gamma} s_4'$, then $s_3 \xrightarrow{\gamma} s_3'$ such that $s_3' \sim_{i-1} s_4'$. We then recognise the following cases.
     (a) $\exists \alpha \in \mathrm{adom}(r). s_2 \stackrel{\alpha}{\Longrightarrow} s_2'$ and $\exists r(\alpha) \xrightarrow{\gamma} v$ such that $s_2' \preccurlyeq^{r,R\oplus[v]} s_3'$. Since $i > 1$, we may deduce that $s_1 \stackrel{\alpha}{\Longrightarrow} s_1'$ such that $s_1' \sim_{i-1} s_2'$. By the induction hypothesis, it follows that $s_1' \preccurlyeq^{r,R\oplus[v]}_{i-2} s_4'$.
     (b) $\exists s_2 \stackrel{\varepsilon}{\Longrightarrow} s_2'$ and $\exists R \xrightarrow{\gamma} R'$ such that $s_2' \preccurlyeq^{r,R'} s_3'$. Since $i > 1$, we may deduce that $s_1 \stackrel{\varepsilon}{\Longrightarrow} s_1'$ such that $s_1' \sim_{i-1} s_2'$. By the induction hypothesis, it follows that $s_1' \preccurlyeq^{r,R'}_{i-2} s_4'$.
     (c) $\gamma \notin \mathrm{arng}(r)$. Then $\exists s_2 \stackrel{\hat{\gamma}}{\Longrightarrow} s_2'$ such that $s_2' \preccurlyeq^{r,R} s_3'$; since $i > 1$, we may deduce that $s_1 \stackrel{\hat{\gamma}}{\Longrightarrow} s_1'$ 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 $R \xrightarrow{\gamma} R'$, then $s_3 \stackrel{\gamma}{\Longrightarrow} s_3'$ such that $s_2 \preccurlyeq^{r,R'} s_3'$. Since $i > 1$, we may deduce that $s_4 \stackrel{\gamma}{\Longrightarrow} s_4'$ 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 $s_1 \xrightarrow{\tau} s_1'$, then (since $i > 0$) $s_2 \xrightarrow{\tau} s_2'$ such that $s_1' \sim_{i-1} s_2'$; hence $s_3 \stackrel{\tau}{\Longrightarrow} s_3'$ such that $s_2' \preccurlyeq^{r,\emptyset} s_3'$; hence (since $i > 0$) $s_4 \stackrel{\tau}{\Longrightarrow} s_4'$ 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 $s_1 \lesssim^r_{i-1} s_4$. □

6.11 Lemma. 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 $t\langle x \mathrel{\rightarrowtail} u\rangle \sim_i t\langle x \mathrel{\rightarrowtail} v\rangle$. 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 $t\langle x \mathrel{\rightarrowtail} u\rangle \sim_{i+1} t\langle x \mathrel{\rightarrowtail} v\rangle$. This can be deduced from the auxiliary result of the previous step, using Propositions 2.3.1 (observing that strict guardedness implies guardedness) and 6.6.1, plus the fact that if x is strictly guarded in t and $t \xrightarrow{\alpha} t'$, 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 behaviour of $\mu x.\, t$ equals that of $t\langle x \rightarrow \mu x.\, t\rangle$. For $i = 0$, the statement is immediate, whereas otherwise it is obtained by applying the result of the previous step to the induction hypothesis. □

A.4 Proofs of Section 7

7.1 Theorem. $\lesssim^r$ satisfies Rule R_{25}.

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

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

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 $\mathbf{1}; t \preccurlyeq^{r,R} u \parallel\!\!\parallel\!\!\parallel v$, 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 $t \preccurlyeq^{r,R_1 \oplus R_2} u \parallel\!\!\parallel\!\!\parallel v$).

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 $u_1 \stackrel{\sigma_1}{\Longrightarrow}$ and $u_2 \stackrel{\sigma_2}{\Longrightarrow} u' \xrightarrow{\surd}$ such that $\sigma_2 \in \mathbf{C}^+$ and $\sigma = \sigma_1\sigma_2$, implying $\mathbf{1} \preccurlyeq^{r,\emptyset} u'$; hence $u_1; (u_2 \parallel\!\!\parallel\!\!\parallel v) \stackrel{\sigma}{\Longrightarrow} u' \parallel\!\!\parallel\!\!\parallel v$ 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 $t \stackrel{\alpha}{\Longrightarrow} t'$, $r(\alpha) \xrightarrow{\gamma} v''$ and $t' \preccurlyeq^{r,[v'']} v'$. Then $\mathbf{1}; t \stackrel{\alpha}{\Longrightarrow} t'$ and $t'\ \rho^{R \oplus [v'']}\ u_2 \parallel\!\!\parallel\!\!\parallel v'$.
  2. $\gamma \notin \mathrm{arng}(r)$ and $t \stackrel{\gamma}{\Longrightarrow} t'$ such that $t' \preccurlyeq^{r,\emptyset} v'$. Then $\mathbf{1}; t \stackrel{\alpha}{\Longrightarrow} t'$ 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 $u; u_2 \stackrel{\gamma}{\Longrightarrow} u'$ such that $\mathbf{1} \preccurlyeq^{r,R'} u'$; hence either $u \xrightarrow{\gamma} u''$ such that $u' = u''; u_2$, in which case $u; (u_2 \parallel\!\!\parallel\!\!\parallel v) \stackrel{\gamma}{\Longrightarrow} u''; (u_2 \parallel\!\!\parallel\!\!\parallel v)$ and $R'\ \kappa\ u''; (u_2 \parallel\!\!\parallel\!\!\parallel v)$, or $u \xrightarrow{\surd}$ and $u_2 \xrightarrow{\gamma} u'$, in which case $u; (u_2 \parallel\!\!\parallel\!\!\parallel v) \stackrel{\gamma}{\Longrightarrow} u' \parallel\!\!\parallel\!\!\parallel v$ 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 bi-root of $\rho^\emptyset$ (neither $a; t$ nor $u_1; (u_2 \parallel\!\!\parallel\!\!\parallel v)$ can do an initial τ-transition). □