Formal Justification of the Rely-Guarantee Paradigm for Shared-Variable Concurrency: A Semantic Approach

F.S. de Boer[1], U. Hannemann[2], and W.-P. de Roever[3]

[1] Utrecht University, Department of Computer Science, Utrecht, The Netherlands,
frankb@cs.uu.nl
[2] Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik II, Kiel, Germany,
{uha,wpr}@informatik.uni-kiel.de

Abstract. This paper introduces a semantic analysis of the Rely-Guarantee (R-G) approach to the compositional verification of shared-variable concurrency. The main contribution is a new completeness proof.

1 Introduction

In the Rely-Guarantee (R-G) approach to the compositional verification of shared-variable concurrency [9, 10, 13] a property of a component process is, in essence, stated as a pair (R, G) consisting of a guarantee property G that the component will satisfy provided the environment of the component satisfies the rely property R. The interpretation of (R, G) has to be carefully defined so as to be non-circular. Informally, a component P satisfies (R, G) if the environment of P violates R *before* component P fails to satisfy G. In this paper we develop a semantic approach to the formal justification of the Rely-Guarantee proof method.

There are two basically different compositional semantic models for shared variable concurrency: *reactive-sequence* semantics [4], and *Aczel-trace* semantics [5]. A reactive sequence of a process P is a sequence of computation steps $\langle \sigma, \sigma' \rangle$ which represent the execution of an atomic action of P in state σ with resulting state σ'. The resulting state of a computation step does not necessarily coincide with the initial state of the subsequent computation step in the sequence. These 'gaps' represent the state-changes induced by the (parallel) environment. Note that thus a reactive sequence abstracts from the *the number and granularity* of the environmental actions. In contrast, an Aczel-trace of a process records all the state-changes (both of the process and its environment) at the level of the atomic actions.

Which of these two semantics of shared-variable concurrency provides a suitable basis for a formal justification of the R-G proof method? A seemingly natural interpretation of R-G specifications in terms of reactive sequences consists of the following.

J. Wing, J. Woodcock, J. Davies (Eds.): FM'99, Vol. II, LNCS 1709, pp. 1245–1265, 1999.

> If the gaps of a reactive sequence satisfy the rely condition then the computation steps of the sequence itself should satisfy the guarantee condition.

However under this interpretation the R-G proof rule for parallel composition will allow the derivation of incorrect R-G specifications. A proper semantic analysis based on reactive sequences can be obtained by the introduction of *stutter steps* as studied in [4]. In fact the addition of arbitrary stutter steps allows one to interpret the gaps of a reactive sequence as stemming from the execution of a single atomic action by the environment. In that case the reactive sequences semantics actually coincides with the Aczel semantics. In the Aczel semantics then we have the following interpretation of R-G specifications.

> If all the atomic environmental actions satisfy the rely condition then the computation steps of the sequence itself should satisfy the guarantee condition.

The main contribution of this paper consists of a new semantic completeness proof of the R-G proof method. An essential aspect of the R-G paradigm is that of finding a characterization of validity of a R-G specification which is non-circular. Indeed, the explicit breaking of cycles in chains of implications between R and G properties associated with the different processes which constitute an (open) network occurs already in Misra and Chandy's formulation of the *Assumption-Commitment* method [12] . As our completeness proof for the R-G paradigm demonstrates, preventing such circularities is straightforward once the appropriate concepts have been defined, and certainly simpler than any method proposed before. As worked out in [1], at an abstract level the breaking of such cycles of dependencies is connected to the use of constructive logics for reasoning about such dependencies, and is related to the use of such logics by Gerard Berry in his work on the semantics of the synchronous language Esterel [2]. The completeness proof for our proposed formalization of the Rely-Guarantee paradigm shows that there is a simple alternative to introducing such logics. The practical relevance of the new formal justification of the R-G paradigm presented in this paper lies in the fact that it determines the exact nature of the rely and guarantee predicates and, consequently, it provides a clear view on the way the R-G proof method is to be applied.

The approach which is followed in this paper is based on the inductive-assertion method [7] which is a methodology for proving state-based transition diagrams correct. It consists of the construction of an *assertion network* by associating with each location of a transition diagram a (state) predicate and with each transition a *verification condition* on the predicates associated with the locations involved; semantically, these predicates are viewed as sets of states. Thus it reduces a statement of correctness of a transition diagram, which consists of a finite number of locations, to a correspondingly finite number of verification conditions on predicates.

The inductive assertion method can be trivially generalized to concurrency by viewing a concurrent transition diagram as the product of its components

and thus reducing it to a sequential system. However this global proof method leads to a number of verification conditions which is exponential in the number of components.

Compositional proof methods in general provide a reduction in the complexity of the number of verification conditions. In this paper we investigate the semantic foundations of the Rely-Guarantee proof method for concurrent systems obtained by sequential and parallel composition from basic transition diagrams. The components of such a concurrent system communicate via shared variables.

Technically, we introduce the new concept of R-G-inductive assertion networks for reasoning about the sequential components, i.e., the transition diagrams, of a concurrent system. By means of compositional proof rules such assertion networks can be used for deducing properties of the whole system.

The paper is organized as follows: we first introduce transition diagrams as our basic control structure and define in section 3 the reactive-sequence semantics. R-G correctness formulae are introduced in section 4 together with our proof system for them. In section 5 we formally define validity of R-G specifications w.r.t. the reactive sequence semantics and give an example why this choice of semantics is not appropriate. On top of the reactive sequence semantics we introduce the Aczel semantics, for which we prove in section 8 completeness of the proof system given in section 4. In section 7 we continue the comparison between Aczel semantics and reactive sequence semantics by extending the latter with stutter steps and proving that this change suffices to get a notion of validity of R-G formulae which is equivalent to the one based on Aczel semantics.

2 Syntax

The basic control construct of our semantical analysis of the Rely-Guarantee (R-G) proof system is that of a transition diagram, i.e., a labeled directed graph where each label denotes an instruction. Given a set of states Σ, an instruction has the following form: a boolean condition $b \in \mathcal{P}(\Sigma)$ followed by a state transformation $f \in \Sigma \to \Sigma$, notation: $b \to f$. The set of states Σ, with typical element σ, is given by $VAR \to VAL$, where VAR , with typical elements $x, y, z, \ldots$, is an infinite set of variables and VAL denotes the underlying domain of values. In the sequel sets of states often will be called *predicates* and (sets of) pairs of states will be called *action predicates*, with typical element *act*, as they reflect the effect of a state transformation (or action) upon the state. We have the following semantic characterization of the variables *involved* in a (action) predicate and a state transformation. This characterization is an approximation of the corresponding syntactic notion of *occurrence* of a variable.

Definition 1. *Let $\bar{x}$ denote a sequence $x_1, \ldots, x_n$ of distinct variables. By $\sigma(\bar{x}) = \sigma'(\bar{x})$ we then abbreviate $\bigwedge_{i=1}^{n} \sigma(x_i) = \sigma'(x_i)$. A predicate $\phi \in \mathcal{P}(\Sigma)$* involves *the variables $x_1, \ldots, x_n$ if*

- $\forall \sigma, \sigma' \in \Sigma.\ \sigma(\bar{x}) = \sigma'(\bar{x}) \Rightarrow (\sigma \in \phi \Leftrightarrow \sigma' \in \phi)$.

This condition expresses that the outcome of ϕ only depends on the variables $x_1, \ldots, x_n$.
Similarly, an action predicate $act \in \mathcal{P}(\Sigma \times \Sigma)$ involves the variables $x_1, \ldots, x_n$ if

- $\forall \langle \sigma_1, \sigma_1' \rangle, \langle \sigma_2, \sigma_2' \rangle \in \mathcal{P}(\Sigma \times \Sigma)$.
  $(\sigma_1(\bar{x}) = \sigma_2(\bar{x}) \wedge \sigma_1'(\bar{x}) = \sigma_2'(\bar{x})) \Rightarrow (\langle \sigma_1, \sigma_1' \rangle \in act \Leftrightarrow \langle \sigma_2, \sigma_2' \rangle \in act)$.

Finally, a function $f \in \Sigma \to \Sigma$ involves the variables $\bar{x}$ if

- $\forall \sigma, \sigma' \in \Sigma.\ \sigma(\bar{x}) = \sigma'(\bar{x}) \Rightarrow f(\sigma)(\bar{x}) = f(\sigma')(\bar{x})$
- $\forall \sigma \in \Sigma, y \notin \bar{x}.\ f(\sigma)(y) = \sigma(y)$

The first condition expresses that if two states σ and σ' agree with respect to the variables $\bar{x}$, then so do their images under f. The second condition expresses that any other variable is not changed by f.

We restrict ourselves to state-transformations and (action) predicates for which there exists a *finite* set of variables which are involved. The set of variables involved in the state-transformation f, (action) predicates ϕ and act, we denote by $var(f)$, $var(\phi)$ and $var(act)$, respectively. For predicate ϕ and action predicate act let $\sigma \models \phi$ denote $\sigma \in \phi$, and $\langle \sigma, \sigma' \rangle \models act$ denote $\langle \sigma, \sigma' \rangle \in act$. By $\models \phi$ (and $\models act$) we denote the validity of ϕ (and act), i.e., for all σ, $\sigma \models \phi$ (and for all $\langle \sigma, \sigma' \rangle$, $\langle \sigma, \sigma' \rangle \models act$).

Given a sequence of distinct variables $\bar{x} = x_1, \ldots, x_n$ and a sequence of values $\bar{v} = v_1, \ldots, v_n$, the state-transformation $(\sigma : \bar{x} \mapsto \bar{v})$ is defined by

$$(\sigma : \bar{x} \mapsto \bar{v})(y) \stackrel{\text{def}}{=} \begin{cases} \sigma(y) & \text{if } y \notin \{x_1, \ldots, x_n\} \\ v_i & \text{if } y = x_i \end{cases}$$

For a sequence of distinct variables $\bar{x} = x_1, \ldots, x_n$, $\exists \bar{x}.\phi$ denotes the set of states σ such that $(\sigma : \bar{x} \mapsto \bar{v}) \in \phi$, for some sequence of values $\bar{v} = v_1, \ldots, v_n$. Similarly, $\exists \bar{x}.act$, act an action predicate, denotes the set of pairs of states $\langle \sigma, \sigma' \rangle$ such that $\langle (\sigma : \bar{x} \mapsto \bar{v}), (\sigma' : \bar{x} \mapsto \bar{v'}) \rangle \in act$, for some sequences of values $\bar{v} = v_1, \ldots, v_n$ and $\bar{v'} = v_1', \ldots, v_n'$. Finally, given a state-transformation f, the state-transformation $\exists \bar{x}.f$ is defined by $\exists \bar{x}.f(\sigma) \stackrel{\text{def}}{=} (f(\sigma) : \bar{x} \mapsto \sigma(\bar{x}))$, where $\sigma(\bar{x})$ denotes the sequence of values $\sigma(x_1), \ldots, \sigma(x_n)$.

We have the following formal definition of a transition diagram.

Definition 2. *A basic transition diagram is a quadruple (L, T, s, t), where L is a finite set of locations l, T is a finite set of transitions $(l, b \to f, l')$, and s and t are the entry and exit locations, respectively, which are different ($s \neq t$). There are no outgoing transitions starting in t.*

A program P is either a basic transition diagram or defined inductively as a sequential composition $P_1; P_2$ or parallel composition $P_1 \parallel P_2$ of two programs P_1 and P_2.

3 Reactive Sequence Semantics

The Rely-Guarantee paradigm aims at specifying both terminating *and nonterminating* computations in a compositional style. We denote termination by the symbol $\surd$.

For the formal definition of reactive sequence semantics as introduced in, e.g., [4], we use the following transition relation.

Definition 3. *For a given basic transition diagram* $P = \langle L, T, s, t\rangle$,

$$l \xrightarrow{\langle\sigma,\sigma'\rangle} l'$$

denotes a transition of P *when for some* $(l, b \to f, l') \in T$ *one has that* $\sigma \models b$ *and* $\sigma' = f(\sigma)$.

The following axiom and rule allow to compute the reflexive transitive closure of this transition relation:

$$l \xrightarrow{\epsilon} l \quad \text{and} \quad \frac{l \xrightarrow{w} l', l' \xrightarrow{w'} l''}{l \xrightarrow{w\cdot w'} l''},$$

where ϵ *denotes the empty sequence, and "*$\cdot$*" the operation of concatenation.*

Given a basic transition diagram P, $l \xrightarrow{w} l'$ thus indicates that starting at l execution of P can generate the sequence of computation steps w arriving at l'. Such a sequence w is called a *reactive sequence*. For a non-empty reactive sequence $w = w'\cdot\langle\sigma,\sigma'\rangle$ we define $laststep(w) \stackrel{\text{def}}{=} \langle\sigma,\sigma'\rangle$. A reactive sequence $w = \langle\sigma_1,\sigma'_1\rangle\langle\sigma_2,\sigma'_2\rangle\cdots\langle\sigma_n,\sigma'_n\rangle$ is called a *connected* sequence if for all $i = 1,\ldots,n-1$ we have that $\sigma'_i = \sigma_{i+1}$. A 'gap' $\langle\sigma'_i,\sigma_{i+1}\rangle$ between two consecutive computation steps $\langle\sigma_i,\sigma'_i\rangle$ and $\langle\sigma_{i+1},\sigma'_{i+1}\rangle$ represents the state-transformation induced by the (parallel) environment. Note that such a gap, therefore, abstracts from the *granularity* of the environment, i.e., the actual number of atomic computation steps performed by the environment.

Definition 4. *For a basic transition diagram* $P = \langle L, T, s, t\rangle$, $l \in L$ *we define* $\mathcal{R}_l[\![P]\!] \stackrel{\text{def}}{=} \{w |\ s \xrightarrow{w} l\}$.

We distinguish sequences which are *terminated* w.r.t. the executing process by ending them with the $\surd$ symbol. *Computations* are either reactive sequences or reactive sequences followed by a $\surd$ symbol. Therefore, if a computation w contains a $\surd$, it is of the form $w'\surd$ with w' a reactive sequence containing no $\surd$ symbol.

Definition 5. *The reactive-sequence semantics* $\mathcal{R}[\![P]\!]$ *of a program* P *is defined as follows: For* $P = \langle L, T, s, t\rangle$ *we define*

$$\mathcal{R}[\![P]\!] \stackrel{\text{def}}{=} \bigcup_{l\in L} \mathcal{R}_l[\![P]\!] \cup \{w\surd |\ w \in \mathcal{R}_t[\![P]\!]\}.$$

For $P = P_1; P_2$ we define

$$\mathcal{R}[\![P]\!] \stackrel{\text{def}}{=} \{w \mid w \in \mathcal{R}'[\![P_1]\!]\} \cup \{w \cdot w' \mid w\surd \in \mathcal{R}[\![P_1]\!] \wedge w' \in \mathcal{R}[\![P_2]\!]\},$$

where $\mathcal{R}'[\![P_1]\!]$ denotes the set of non-terminated sequences of P_1, that is, those sequences not ending with $\surd$. Finally, for $P = P_1 \parallel P_2$ we define

$$\mathcal{R}[\![P]\!] \stackrel{\text{def}}{=} \{w \mid w \in w_1 \tilde{\|} w_2, w_1 \in \mathcal{R}[\![P_1]\!], w_2 \in \mathcal{R}[\![P_2]\!]\},$$

where $w_1 \tilde{\|} w_2$ denotes the set of all interleavings of w_1 and w_2, ending in $\surd$ if and only if both w_1 and w_2 end in $\surd$.

The semantics $\mathcal{R}[\![P]\!]$ contains all the finite prefixes of all the computations of P, including the *non-terminating* computations. Recall from the introduction that a process P satisfies (R, G) provided P's environment violates R *before* P violates G, i.e., *at any stage of an on-going computation* P's actions should satisfy G *as long as* R *remains satisfied by* P*'s environment.* This is mathematically expressed by requiring (R, G) to be satisfied by all prefixes of a computation of P.

So how does one characterize the semantics of programs in which the this process of parallel composition with new environments has come to an end, i.e., the semantics of a closed system? This is done by considering only reactive sequences in which the gaps are "closed", i.e., by considering the subset of connected sequences.

4 The Rely-Guarantee Proof Method

In this section we first give an intuitive definition of Rely-Guarantee correctness formulae and their interpretation and then present a proof system for this type of correctness formula that is fairly standard as far as the composition rules are concerned [15]. For correctness formulae that reason about basic transition diagrams we adapt Floyd's inductive assertion network method [7] to the additional requirements of the R-G method.

Definition 6. *Let pre and post be predicates denoting sets of states, rely and guar be action predicates, and P be a program, then $\langle rely, guar\rangle : \{pre\}\ P\ \{post\}$ is called an R-G correctness formula.*

Traditionally, *pre* and *post* impose conditions upon the initial, respectively, final state of a computation, whereas *rely* and *guar* impose conditions upon environmental transitions, respectively, transitions of the process itself. This is captured by the following intuitive characterization of validity of an R-G formula:

Whenever

1) P is invoked in an initial state which satisfies *pre*, and
2) the environment satisfies *rely*,

then

3) any transition of P satisfies *guar*, and
4) if a computation terminates, its final state satisfies *post*.

We generalize Floyd's method to the additional requirements of R-G formulae and define for $P = \langle L, T, s, t\rangle$ an R-G-inductive assertion network $\mathcal{Q}(rely, guar) : L \to \mathcal{P}(\Sigma)$, i.e., we associate with each location l a predicate $\mathcal{Q}_l$ as follows:

Definition 7 (R-G-inductive assertion networks). *An assertion network $\mathcal{Q}$ is R-G-inductive w.r.t. rely and guar for $P = \langle L, T, s, t\rangle$ if:*

- *For every $(l, b \to f, l') \in T$ and state σ: if $\sigma \models \mathcal{Q}_l \wedge b$ then $\langle \sigma, f(\sigma)\rangle \models guar$ and $f(\sigma) \models \mathcal{Q}_{l'}$.*
- *For every $l \in L$ and states σ and σ': if $\sigma \models \mathcal{Q}_l$ and $\langle \sigma, \sigma'\rangle \models rely$ then $\sigma' \models \mathcal{Q}_l$.*

We abbreviate that $\mathcal{Q}$ is an R-G-inductive assertion network w.r.t. *rely* and *guar* for P by $\mathcal{Q}(rely, guar) \vdash P$. We have the following rule for deriving R-G specifications about basic transition diagrams.

Rule 8 (Basic diagram rule) *For $P = \langle L, T, s, t\rangle$:*

$$\frac{\mathcal{Q}(rely, guar) \vdash P}{\langle rely, guar\rangle : \{\mathcal{Q}_s\}\ P\ \{\mathcal{Q}_t\}}$$

The following rules are standard.

Rule 9 (Sequential composition rule)

$$\frac{\langle rely, guar\rangle : \{\phi\}\ P_1\ \{\chi\},\ \langle rely, guar\rangle : \{\chi\}\ P_2\ \{\psi\}}{\langle rely, guar\rangle : \{\phi\}\ P_1; P_2\ \{\psi\}}$$

Rule 10 (Parallel composition rule)

$$\frac{\begin{array}{c}\models rely \vee guar_1 \to rely_2 \\ \models rely \vee guar_2 \to rely_1 \\ \models guar_1 \vee guar_2 \to guar \\ \langle rely_i, guar_i\rangle : \{pre\}\ P_i\ \{post_i\},\ i = 1, 2\end{array}}{\langle rely, guar\rangle : \{pre\}\ P_1 \| P_2\ \{post_1 \wedge post_2\}}$$

Rule 11 (Consequence rule)

$$\frac{\begin{array}{c}\langle rely, guar\rangle : \{\phi\}\ P\ \{\psi\} \\ \models \phi_1 \to \phi,\ \models \psi \to \psi_1, \\ \models rely_1 \to rely, \models guar \to guar_1\end{array}}{\langle rely_1, guar_1\rangle : \{\phi_1\}\ P\ \{\psi_1\}}$$

Definition 12. *A set of program variables $\bar{z} = z_1, \ldots, z_n$ is called a set of auxiliary variables of a program P if:*

- *For any boolean condition b of P we have $\bar{z} \cap var(b) = \emptyset$, and*
- *any state transformation of P can be written as $f \circ g$, i.e., a composition of state-transformations f and g, such that $\bar{z} \cap var(f) = \emptyset$, and the write variables of g, i.e, those variables x such that $g(\sigma)(x) \neq \sigma(x)$, for some state σ, are among $\bar{z}$.*

We have the following rule for deleting auxiliary variables:

Rule 13 (Auxiliary variables rule)

$$\frac{\langle rely, guar\rangle : \{\phi\}\ P'\ \{\psi\}}{\langle \exists \bar{z}.rely, guar\rangle : \{\exists \bar{z}.\phi\}\ P\ \{\psi\}},$$

where $\bar{z}$ is a set of auxiliary variables of P', $guar$ and ψ do not involve $\bar{z}$, and P is obtained from P' by replacing every state transformation f in P' by $\exists \bar{z}.f$.

Finally, how does one reason about closed programs? This is done by requiring *rely* to be *id*, the identity on states.

Derivability of an R-G formula $\langle rely, guar\rangle : \{\phi\}\ P\ \{\psi\}$ in this proof system is expressed by

$$\vdash \langle rely, guar\rangle : \{\phi\}\ P\ \{\psi\}.$$

5 R-G Validity w.r.t. Reactive Sequences Semantics

In order to define the validity of a R-G specification $\langle rely, guar\rangle : \{\phi\}P\{\psi\}$ we have first to determine the exact meaning of the precondition ϕ and the postcondition ψ: Are these predicates referring to the initial and final state of P itself or of the complete system (which includes the environment of P)? Following the literature we choose the latter option. Therefore we define the validity of a R-G specification for P in terms of a triple consisting of an initial (i.e., w.r.t. the complete system) state σ, a reactive sequence w of P, which records the sequence of computation steps of P, and a final state σ', which is final under the assumption that the environment has terminated as well. Whereas for terminated computations σ' is the final state of the complete system, we can interpret it as the "current" state for non-terminating computations, i.e., the last observation point at hand.

We define for a reactive sequence w and states σ, σ' the *complement* of w with respect to initial state σ and final state σ', denoted by $\overline{\langle \sigma, w, \sigma'\rangle}$, as follows:

Definition 14. *We define*

$$\overline{\langle \sigma, \epsilon, \sigma'\rangle} \stackrel{\text{def}}{=} \langle \sigma, \sigma'\rangle,$$
$$\overline{\langle \sigma,\ \langle \sigma_1, \sigma_2\rangle \cdot w, \sigma'\rangle} \stackrel{\text{def}}{=} \langle \sigma, \sigma_1\rangle \cdot \overline{\langle \sigma_2,\ w, \sigma'\rangle}.$$

The complement of a reactive sequence w with respect to a given initial state σ and final state σ' thus specifies the behavior of the environment.

Definition 15. *For a reactive sequence $w = \langle \sigma_1, \sigma_1' \rangle \cdots \langle \sigma_n, \sigma_n' \rangle$, $w \models act$ indicates that $\langle \sigma_i, \sigma_i' \rangle \models act$, $i = 1, \ldots, n$, (and $w\surd \models act$ indicates that $w \models act$).*

Now we are sufficiently equipped to introduce the following notion of validity of R-G specifications.

Definition 16 (R-Validity of R-G specifications). *We define*

$$\models_R \langle rely, guar \rangle : \{\phi\}\ P\ \{\psi\}$$

by

> *for all $w \in \mathcal{R}[\![P]\!]$, states σ and σ', if $\sigma \models \phi$ and $\overline{\langle \sigma, w, \sigma' \rangle} \models rely$ then $w \models guar$ and $w = w'\surd$, for some w', implies $\sigma' \models \psi$.*

Intuitively, a R-G specification $\langle rely, guar \rangle : \{\phi\}\ P\ \{\psi\}$ is R-valid if for every reactive sequence w of P, initial state σ and final state σ' (of the parallel composition of P with its environment) the following holds: if the initial state σ satisfies ϕ and all the steps of the environment as specified by $\overline{\langle \sigma, w, \sigma' \rangle}$ satisfy *rely* then all the steps of w satisfy *guar* and upon termination the final state σ' satisfies ψ.

Example 1. We have the following *counter-example* to the soundness of the parallel composition rule with respect to the notion of R-validity above: It is not difficult to check that

$$\models_R \langle x' = x + 1, x' = x + 1 \rangle : \{x = 0\} x := x + 1 \{x = 3\}.$$

By an application of the parallel composition rule to $x := x + 1 \parallel x := x + 1$, where both assignments $x := x + 1$ are specified as above, we then would derive

$$\langle true, x' = x + 1 \rangle : \{x = 0\} x := x + 1 \parallel x := x + 1 \{x = 3\}$$

which is clearly not R-valid.

(Here $x := x + 1$ abbreviates the transition diagram $\langle \{s, t\}, \{(s, true \rightarrow f, t)\}, s, t \rangle$, where f increments x by 1.)

In the full paper we show that soundness of the parallel composition rule with respect to this notion of validity requires all *rely*-predicate to be *transitive*, i.e., that $\langle \sigma, \sigma' \rangle \models rely$ and $\langle \sigma', \sigma'' \rangle \models rely$ imply $\langle \sigma, \sigma'' \rangle \models rely$.

This observation motivates our next section where we give a different interpretation of R-G specifications in terms of Aczel-traces. These Aczel-traces will provide more detailed information about environmental steps.

6 Aczel Semantics

An Aczel-trace is a connected sequence of process-indexed state pairs. It can thus be seen as the extension of connected reactive sequence in which every atomic action contains as additional information an identifier which represents the executing process.

We assume to have a set Id of *process identifiers* with typical element $I_1, I_2, \ldots$. The complement of a set of identifiers $V \subseteq Id$ is denoted by $\overline{V} \stackrel{\text{def}}{=} Id \setminus V$.

Definition 17. *A process-indexed state pair is a triple $\langle \sigma, I, \sigma' \rangle \in \Sigma \times Id \times \Sigma$. An Aczel-trace π is a non-empty connected sequence of process-indexed state pairs, that might end with a $\surd$-symbol.*
For an Aczel-trace π we define $first(\pi)$ and $last(\pi)$ by the first and last state of the sequence, respectively: $first(\langle \sigma, I, \sigma' \rangle \cdot \pi') = \sigma$ and $last(\pi' \cdot \langle \sigma, I, \sigma' \rangle) = \sigma'$. ($last(\pi\surd) = last(\pi)$).

In order to define the set of Aczel-traces of a program in terms of its reactive-sequence semantics we introduce the following projection operation on Aczel-traces.

Definition 18. *Let $V \subseteq Id$ be a set of identifiers.*

$$\begin{aligned}
&\epsilon[V] \stackrel{\text{def}}{=} \epsilon,\\
&(\langle \sigma, I, \sigma' \rangle \cdot \pi)[V] \stackrel{\text{def}}{=} \pi[V], \text{ if } I \notin V,\\
&(\langle \sigma, I, \sigma' \rangle \cdot \pi)[V] \stackrel{\text{def}}{=} \langle \sigma, \sigma' \rangle \cdot \pi[V], \text{ if } I \in V,\\
&\pi\surd[V] \stackrel{\text{def}}{=} \pi[V]\surd.
\end{aligned}$$

We define the Aczel semantics of a program P parametric with respect to a set of identifiers V. The elements of V are used to identify the transitions of P. Thus we can extract a reactive sequence of P out of an Aczel-trace by projecting onto this set of identifiers. Within Aczel-traces, the purpose of these identifiers is to distinguish between steps of the process and steps of the environment.

Definition 19. *For $P = \langle L, T, s, t \rangle$ a basic transition diagram, $l \in L$, and $V \subseteq Id$, we define*

$$\mathcal{A}cz_V^l [\![P]\!] \stackrel{\text{def}}{=} \{\pi \,|\, \pi[V] \in \mathcal{R}_l [\![P]\!]\}.$$

By $\mathcal{A}cz_V [\![P]\!]$ then we denote $\mathcal{A}cz_V^t [\![P]\!]$. For composed systems P we define

$$\mathcal{A}cz_V [\![P]\!] \stackrel{\text{def}}{=} \{\pi \,|\, \pi[V] \in \mathcal{R} [\![P]\!]\}.$$

The proof of the following proposition is straightforward and therefore omitted.

Proposition 20. *Let V_1 and V_2 be disjoint sets of identifiers. We have for every $P = P_1 \parallel P_2$*

$$\mathcal{A}cz_{V_1} [\![P_1]\!] \cap \mathcal{A}cz_{V_2} [\![P_2]\!] \subseteq \mathcal{A}cz_V [\![P]\!],$$

with $V = V_1 \cup V_2$. Note that in general the converse does not hold.

We have the following interpretation of R-G specifications.

Definition 21 (Aczel-Validity of R-G specifications). *We define*

$$\models_A \langle rely, guar\rangle : \{\phi\}\ P\ \{\psi\}$$

by

For all sets of identifiers V and $\pi \in \mathcal{A}cz_V[\![P]\!]$ if $first(\pi) \models \phi$ and $\pi[\overline{V}] \models rely$ then $\pi[V] \models guar$ and $\pi = \pi'\surd$ implies $last(\pi) \models \psi$.

The R-G method as presented above is sound with respect to the Aczel-trace semantics, for the soundness proof of the basic diagram rule we refer to the full paper. For the other rules detailed proofs in the Aczel-trace set-up are given in [15].

The main difference between this notion of validity and the one based on reactive sequences is that now *every atomic* computation step of the environment has to satisfy the rely condition. Consequently, for Example 1 we have

$$\not\models_A \langle x' = x + 1, x' = x + 1\rangle : \{x = 0\}x := x + 1\{x = 3\},$$

since there is an arbitrary number of environmental steps possible.

7 Reactive Sequences Reconsidered

As observed above the reactive sequences semantics $\mathcal{R}$ does not provide a correct interpretation of R-G specifications. More precisely, it requires the predicates $rely_1$ and $rely_2$ in the parallel composition rule to be transitive. However, we can obtain such a correct interpretation of R-G specifications by the introduction of arbitrary *stutter steps* of the form $\langle\sigma, \sigma\rangle$.

Definition 22. *Let $\mathcal{R}_\tau[\![P]\!]$ be the smallest set containing $\mathcal{R}[\![P]\!]$ which satisfies the following:*

$$w_1 \cdot w_2 \in \mathcal{R}_\tau[\![P]\!] \text{ implies } w_1 \cdot \langle\sigma, \sigma\rangle \cdot w_2 \in \mathcal{R}_\tau[\![P]\!].$$

This abstraction operation is required in order to obtain a *fully abstract* semantics (see [4, 3, 11]). We observe that the corresponding notion of validity, which we denote by $\models_{R_\tau}$, requires the *guar* predicate to be *reflexive*, i.e. $\langle\sigma, \sigma\rangle \models guar$, for every state σ. However, given this restriction we do have that the two different notions of validity $\models_A$ and $\models_{R_\tau}$ coincide.

Theorem 23. *Let $\langle rely, guar\rangle : \{\phi\}P\{\psi\}$ be such that guar is reflexive. Then*

$$\models_A \langle rely, guar\rangle : \{\phi\}P\{\psi\} \text{ if and only if } \models_{R_\tau} \langle rely, guar\rangle : \{\phi\}P\{\psi\}.$$

Proof. Let $\models_A \langle rely, guar\rangle : \underline{\{\phi\}P\{\psi\}}$ and $w \in \mathcal{R}_\tau[\![P]\!]$. Furthermore let σ and σ' be such that $\sigma \models \phi$, $\langle \sigma, w, \sigma'\rangle \models rely$. Then the requirements of $\models_{R_\tau}$ are satisfied because of the existence of a corresponding $\pi \in \mathcal{A}cz_V[\![P]\!]$, for any (non-empty) V. Formally, we obtain such a corresponding π by defining the Aczel-trace $A(\sigma, w, \sigma')$ by induction on the length of w. Let $E \notin V$ and $I \in V$. Then

$$\begin{aligned} A(\sigma, \epsilon, \sigma') &\stackrel{\text{def}}{=} \langle \sigma, E, \sigma'\rangle, \\ A(\sigma, \langle \sigma_1, \sigma_2\rangle \cdot w, \sigma') &\stackrel{\text{def}}{=} \langle \sigma, E, \sigma_1\rangle \cdot \langle \sigma_1, I, \sigma_2\rangle \cdot A(\sigma_2, w, \sigma'). \end{aligned}$$

Conversely, let $\models_{R_\tau} \langle rely, guar\rangle : \{\phi\}P\{\psi\}$ and $\pi = \langle \sigma_1, I_1, \sigma_2\rangle \cdots \langle \sigma_n, I_n, \sigma_{n+1}\rangle \in \mathcal{A}cz_V[\![P]\!]$ such that $\sigma_1 \models \phi$ and $\langle \sigma_k, \sigma_{k+1}\rangle \models rely$, for $I_k \notin V$. Then the requirements of $\models_A$ are satisfied because of the existence of a corresponding $w \in \mathcal{R}_\tau[\![P]\!]$. Formally, we define $R(\pi)$ by induction on the length of π:

$$\begin{aligned} R(\epsilon) &\stackrel{\text{def}}{=} \epsilon, \\ R(\langle \sigma_1, I_1, \sigma_2\rangle \cdot \pi) &\stackrel{\text{def}}{=} \begin{cases} \langle \sigma_1, \sigma_2\rangle \cdot R(\pi) & I_1 \in V \\ \langle \sigma_2, \sigma_2\rangle \cdot R(\pi) & I_1 \notin V, \end{cases} \end{aligned}$$

and use $R(\pi) \in \mathcal{R}_\tau[\![P]\!]$ as reactive sequence corresponding to π to prove that *guar* and ψ hold in their respective (pairs of) states. Note that thus the insertion of stutter steps is used to obtain the 'gaps' corresponding to the environmental steps in π, providing extra observation points.

8 Completeness

This section presents the completeness proof for our proof system and constitutes the very justification of the paper. We have the following main theorem (the remainder of this section is devoted to its proof).

Theorem 24. *The proof system presented in section 4 is (relative) complete w.r.t. the Aczel-trace semantics, i.e.,*

$$\models_A \langle rely, guar\rangle : \{\phi\}\ P\ \{\psi\} \text{ implies } \vdash \langle rely, guar\rangle : \{\phi\}\ P\ \{\psi\}.$$

We prove the derivability of an Aczel-valid R-G specification by induction on the structure of the program P.

Basic case

Given a valid R-G specification $\models_A \langle rely, guar\rangle : \{\phi\}\ P\ \{\psi\}$, with $P = \langle L, T, s, t\rangle$ a basic transition diagram, we associate with every location l of P the *strongest postcondition* $SP_l(\phi, rely, P)$. The resulting network we denote by $\mathcal{SP}$. Intuitively, a state σ belongs to $SP_l(\phi, rely, P)$ if there is a computation of P *together with its environment* that reaches location l of P, starting in a state satisfying ϕ, such that all environment steps satisfy *rely*.

Definition 25. *For $P = \langle L, T, s, t\rangle$ we define*

$$\sigma \models SP_l(\phi, rely, P)$$

by

$\sigma \models \phi$ *(in case* l *equals* s*) or* $first(\pi) \models \phi$ *and* $\pi[\overline{V}] \models rely$*, for some set* V *of process identifiers and some* $\pi \in \mathcal{A}cz_V^l [\![P]\!]$*, with* $last(\pi) = \sigma$*.*

Note that any state σ' which can be reached from a state σ which satisfies $SP_l(\phi, rely, P)$ by a sequence of $rely$-steps also satisfies $SP_l(\phi, rely, P)$, because any computation sequence of P together with its environment that reaches location l of P in state σ can be extended to a similar sequence reaching σ'. Hence $SP_l(\phi, rely, P)$ is invariant under $rely$.

We also need a characterization of the computation steps of a program P. This is given by the *strongest guarantee* $SG(\phi, rely, P)$, an action predicate describing those transitions of P which are actually executed by P in some computation, provided ϕ is satisfied initially, and every environment transition satisfies $rely$.

Definition 26. *Let* P *be an arbitrary program. We define*

$$\langle \sigma, \sigma' \rangle \models SG(\phi, rely, P)$$

by

$first(\pi) \models \phi$ *and* $\pi[\overline{V}] \models rely$*, for some set* V *of process identifiers and* $\pi \in \mathcal{A}cz_V [\![P]\!]$*, with* $\langle \sigma, \sigma' \rangle = laststep(\pi[V])$*.*

The following basic properties of SP_l and SG follow immediately from their definitions.

Lemma 27. *For* P *a basic transition diagram we have*

i) $\models_A \langle rely, SG(\phi, rely, P) \rangle : \{\phi\}\ P\ \{SP_t(\phi, rely, P)\}$.
ii) $\models_A \langle rely, guar \rangle : \{\phi\}\ P\ \{\psi\}$ *implies*
  a) $\models SP_t(\phi, rely, P) \rightarrow \psi$.
  b) $\models SG(\phi, rely, P) \rightarrow guar$.
iii) $\models \phi \rightarrow SP_s(\phi, rely, P)$.

Moreover, we have the following lemma.

Lemma 28. *Given a basic transition diagram* P*,* $\mathcal{SP}$ *is an R-G-inductive assertion network w.r.t.* $rely$ *and* $SG(\phi, rely, P)$*.*

Proof. Let $l \in L$ and $\sigma \models SP_l(\phi, rely, P)$. So, for some set V of process identifiers, there exists π such that $\pi \in \mathcal{A}cz_V^l [\![P]\!]$, $first(\pi) \models \phi$, $\pi[\overline{V}] \models rely$ and $last(\pi) = \sigma$.

- Let $\sigma \models b$ and $(l, b \rightarrow f, l') \in T$. By executing $(l, b \rightarrow f, l')$ we reach l' with $\sigma' = f(\sigma)$. We first prove that $\sigma' \models SP_{l'}(\phi, rely, P)$. Without loss of generality we may assume that V is non-empty. Let $I \in V$. Since $\pi \in \mathcal{A}cz_V^l [\![P]\!]$ we get that $\pi' = \pi \cdot \langle \sigma, I, \sigma' \rangle \in \mathcal{A}cz_V^{l'} [\![P]\!]$. We have that $first(\pi) = first(\pi') \models \phi$ (note that π is non-empty). Moreover, $\pi[\overline{V}] \models rely$ and $\pi[\overline{V}] = \pi'[\overline{V}]$. Thus, $\pi'[\overline{V}] \models rely$. Obviously we have $last(\pi') = \sigma'$ and therefore $\sigma' \models SP_{l'}(\phi, rely, P)$. Additionally, since $\langle \sigma, \sigma' \rangle = laststep(\pi'[V])$ we derive that $\langle \sigma, \sigma' \rangle \models SG(\phi, rely, P)$.

– Next let $\langle \sigma, \sigma' \rangle \models rely$. We have for $I \notin V$ that $\pi' = \pi \cdot \langle \sigma, I, \sigma' \rangle \in \mathcal{A}cz_V^l [\![P]\!]$. Again we have that $first(\pi) = first(\pi') \models \phi$. Since $\pi[\overline{V}] \models rely$, $\langle \sigma, \sigma' \rangle \models rely$ and $\pi'[\overline{V}] = \pi[\overline{V}] \cdot \langle \sigma, \sigma' \rangle$ we conclude that $\pi'[\overline{V}] \models rely$. Finally, $last(\pi') = \sigma'$. Thus, $\sigma' \models SP_l(\phi, rely, P)$.

By our basic rule 8 we thus derive that

$$\vdash \langle rely, SG(\phi, rely, P) \rangle : \{SP_s(\phi, rely, P)\}\ P\ \{SP_t(\phi, rely, P)\}.$$

Since by Lemma 27

– $\models \phi \rightarrow SP_s(\phi, rely, P)$,
– $\models SP_t(\phi, rely, P) \rightarrow \psi$, and
– $\models SG(\phi, rely, P') \rightarrow guar$

hold, we derive by the consequence rule

$$\vdash \langle rely, guar \rangle : \{\phi\}\ P\ \{\psi\}.$$

Composed programs
Next we consider the remaining cases $P = P_1; P_2$ and $P = P_1 \parallel P_2$. First we generalize definition 25.

Definition 29. *We define for every system P,*

$$\sigma \models SP(\phi, rely, P)$$

if

> *$first(\pi) \models \phi$ and $\pi[\overline{V}] \models rely$, for some set V of process identifiers and some $\pi\surd \in \mathcal{A}cz_V [\![P]\!]$, with $last(\pi) = \sigma$.*

Note that for $P = (L, T, s, t)$ a basic transition diagram $SP(\phi, rely, P) = SP_t(\phi, rely, P)$. The basic properties of Lemma 27 carry over to the general case.

Lemma 30. *For every system P we have*

i) $\models_A \langle rely, SG(\phi, rely, P) \rangle : \{\phi\}\ P\ \{SP(\phi, rely, P)\}$.
ii) $\models_A \langle rely, guar \rangle : \{\phi\}\ P\ \{\psi\}$ *implies*
  a) $\models SP(\phi, rely, P) \rightarrow \psi$.
  b) $\models SG(\phi, rely, P) \rightarrow guar$.

Sequential composition
Now consider the case of sequential composition. Let

$$\models_A \langle rely, guar \rangle :\ \{\phi\} P_1; P_2 \{\psi\}.$$

By the induction hypothesis we thus obtain

$$\vdash \langle rely, SG(\phi, rely, P_1) \rangle :\ \{\phi\} P_1 \{SP(\phi, rely, P_1)\}$$

and

$$\vdash \langle rely, SG(\phi', rely, P_2)\rangle : \{\phi'\}P_2\{SP(\phi', rely, P_2)\},$$

where $\phi' = SP(\phi, rely, P_1)$. Furthermore,

$$\models_A \langle rely, guar\rangle : \{\phi\}P_1; P_2\{\psi\}$$

implies

$$\models_A \langle rely, guar\rangle : \{\phi\}P_1\{SP(\phi, rely, P_1)\}$$

and

$$\models_A \langle rely, guar\rangle : \{SP(\phi, rely, P_1)\}P_2\{\psi\}.$$

Using the above lemma we thus obtain by the consequence rule

$$\vdash \langle rely, guar\rangle : \{\phi\}P_1\{SP(\phi, rely, P_1)\}$$

and

$$\vdash \langle rely, guar\rangle : \{SP(\phi, rely, P_1)\}P_2\{\psi\}.$$

An application of the rule for sequential composition concludes the proof.

Parallel composition

We have now arrived at the most interesting case $P = P_1 \| P_2$. Let

$$\models_A \langle rely, guar\rangle : \{\phi\}P_1 \| P_2\{\psi\}.$$

Our task is to construct predicates that fit the parallel composition rule 10. In particular we have to define predicates $rely_i, guar_i, pre, post_i$, $i = 1, 2$, such that for some augmentation P_i' of P_i with auxiliary variables the R-G specifications

$$\models_A \langle rely_i, guar_i\rangle : \{pre\}\ P_i'\ \{post_i\},\ i = 1, 2,$$

and the corresponding side conditions hold.

In order to define such predicates we introduce *histories*.

Definition 31. *A history θ is a sequence of indexed states (I, σ), with $I \in Id$.*

An indexed state (I, σ) indicates that the process I is active in state σ.

We assume given a set of history variables $HVAR \subseteq VAR$ with typical element h. For h a history variable, $\sigma(h)$ is a history.

Our next step is to augment every transition of $P_1 \| P_2$ with a corresponding update to the fresh history variable h (i.e., h does not occur in P nor in the given predicates *rely*, *guar*, ϕ, and ψ). This history variable h records the history of P, i.e., the sequence of state changes of process P *together with its environment*, plus the active components responsible for these changes. Without loss of generality we may assume that P_1 and P_2 are two distinct process identifiers. We then transform each transition $(l, b \to f, l')$ of a constituent of P_i to $(l, b \to f \circ g, l')$, where $g \stackrel{\text{def}}{=} (\sigma : h \mapsto h \cdot (P_i, \sigma))$, i.e., $g(\sigma)$ is like σ, except for the value of h which is extended by (P_i, σ). This augmented version of P_i will be denoted by P_i'.

Note that in the augmented process $P' = P_1' \| P_2'$ boolean conditions do not involve the history variable h, and that h does not occur in assignments to non-history variables. I.e., the history variable h is an *auxiliary variable* which does not influence the flow-of-control of a process.

We have to ensure, *in order to have the complete computation history recorded in* h, that every possible environmental action should update the history variable correctly. I.e., we should prevent that some process is setting, e.g., $h := \epsilon$, by formulating additional requirements upon *rely*; also we change the given pre-condition ϕ to ensure that initially h denotes the empty sequence.

Definition 32. *We define*

- $\langle \sigma, \sigma' \rangle \models rely'$ *if and only if* $\langle \sigma, \sigma' \rangle \models rely$ *and* $\sigma'(h) = \sigma(h) \cdot (E, \sigma)$
- $\sigma \models \phi'$ *if and only if* $\sigma \models \phi$ *and* $\sigma(h) = \epsilon$,

where $E \in Id$ *is a process identifier distinct from* P_1 *and* P_2, *representing "the environment".*

It is straightforward to prove that

$$\models_A \langle rely, guar \rangle : \{\phi\} P_1 \| P_2 \{\psi\}$$

implies

$$\models_A \langle rely', guar \rangle : \{\phi'\} P_1' \| P_2' \{\psi\}.$$

Moreover, we introduce the following rely condition e_i which ensures a correct update of the history variable h by the environment of P_i' when executed in the context $P_1' \parallel P_2'$. Note that the environment of P_i' in the context of $P_1' \parallel P_2'$ consists of the common environment of P_1' and P_2' and the other component P_j', $i \neq j$.

Definition 33. *Let for* $i = 1, 2$,

$$\langle \sigma, \sigma' \rangle \models e_i$$

be defined by

$$\sigma'(h) = \sigma(h) \cdot (E, \sigma) \text{ or } \sigma'(h) = \sigma(h) \cdot (P_j, \sigma),$$

where $i \neq j (\in \{1, 2\})$.

We are now in a position to define the predicates that will satisfy the requirements of the parallel composition rule.

Definition 34. *We define for* $i = 1, 2$ *the following predicates*

- $rely_i \stackrel{\text{def}}{=} rely' \vee SG(\phi', e_j, P_j')$ $(i \neq j)$;
- $post_i \stackrel{\text{def}}{=} SP(\phi', rely_i, P_i')$;
- $guar_i \stackrel{\text{def}}{=} SG(\phi', rely_i, P_i')$.

The predicate $rely_i$ is intended to specify the steps of the environment of P_i' in the context of $P_1' \parallel P_2'$. The computation steps of the common environment of P_1' and P_2' are specified by the action predicate $rely'$ whereas the computation steps of the other component are specified by the action predicate $SG(\phi', e_j, P_j')$ which states the existence of a corresponding computation of P_j' in which the environment correctly updates the history variable h.

By Lemma 30 we have for $i = 1, 2$

$$\models_A \langle rely_i, guar_i \rangle : \{\phi'\}\ P_i'\ \{post_i\}.$$

By the induction hypothesis we thus have

$$\vdash \langle rely_i, guar_i \rangle : \{\phi'\} P_i' \{post_i\}.$$

We therefore now prove the corresponding requirements of the parallel composition rule.

Lemma 35. *We have for $i, j = 1, 2$ and $i \neq j$*

$$\models rely' \vee guar_i \rightarrow rely_j,$$

and

$$\models guar_1 \vee guar_2 \rightarrow guar.$$

Proof. The validity of the implication

$$rely' \vee guar_i \rightarrow rely_j$$

follows from the validity of the implication

$$SG(\phi', rely_i, P_i') \rightarrow SG(\phi', e_i, P_i').$$

Validity of this latter implication in turn follows from the validity of the implication $rely_i \rightarrow e_i$. Let $\langle \sigma, \sigma' \rangle \models rely_i$. In case $\langle \sigma, \sigma' \rangle \models rely'$, by definition of $rely'$, we have that $\sigma'(h) = \sigma(h) \cdot (E, \sigma)$, otherwise $\langle \sigma, \sigma' \rangle \models SG(\phi', e_j, P_j')$, and so we have by definition of SG and the construction of P_j', that $\sigma'(h) = \sigma(h) \cdot (P_j, \sigma)$.

In order to prove the validity of the implication

$$guar_1 \vee guar_2 \rightarrow guar$$

let $\langle \sigma, \sigma' \rangle \in guar_i$. By definition of $guar_i$ there exists

$$\pi = \langle \sigma_1, I_1, \sigma_2 \rangle \cdots \langle \sigma_n, I_n, \sigma_{n+1} \rangle \in \mathcal{A}cz_V \left[\!\left[P_i' \right]\!\right],$$

for some set of process identifiers V such that $\sigma_1 \models \phi'$, $\sigma_n = \sigma$, $\sigma_{n+1} = \sigma'$, and $\langle \sigma_k, \sigma_{k+1} \rangle \models rely_i$, whenever $I_k \notin V$. Note that by definition of $rely_i$ and construction of P_j', $(i \neq j)$, $I_k \notin V$ implies either $\sigma_{k+1}(h) = \sigma_k(h) \cdot (E, \sigma_k)$ or $\sigma_{k+1}(h) = \sigma_k(h) \cdot (P_j, \sigma_k)$. Moreover, for $I_k \in V$ we have by construction of P_i' that $\sigma_{k+1}(h) = \sigma_k(h) \cdot (P_i, \sigma_k)$. Thus we may assume without loss of generality

that $\sigma_{k+1}(h) = \sigma_k(h) \cdot (I_k, \sigma_k)$, $k = 1, \ldots, n$ (simply rename the identifiers I_k accordingly). Since, $\sigma_1(h) = \epsilon$, we derive by a straightforward induction that $\sigma_{k+1}(h) = (I_1, \sigma_1) \cdots (I_k, \sigma_k)$, $k = 1, \ldots, n$.

Either there is a last P_j step in the Aczel trace π or there isn't one. If there is no such step in π then also $\pi \in \mathcal{A}cz_{V \cup W} [\![P_1' \| P_2']\!]$, for any set W of identifiers, because there are no P_j steps in π. Otherwise, let $\langle \sigma_l, P_j, \sigma_{l+1} \rangle$ be the last P_j $(i \neq j)$ step of the Aczel-trace π. We have that $\langle \sigma_l, \sigma_{l+1} \rangle \models SG(\phi', e_j, P_j')$. By definition of $SG(\phi', e_j, P_j')$ there exists

$$\pi' = \langle \sigma_1', I_1', \sigma_2' \rangle \cdots \langle \sigma_m', I_m, \sigma_{m+1}' \rangle \in \mathcal{A}cz_W [\![P_j']\!],$$

for some set of process identifiers W such that $\sigma_1' \models \phi'$, $\sigma_m' = \sigma_l$, $\sigma_{m+1}' = \sigma_{l+1}$, and $\langle \sigma_k', \sigma_{k+1}' \rangle \models e_j$, whenever $I_k' \notin W$. By definition of e_j and the construction of P_j', in a similar manner as argued above, we may assume without loss of generality that $\sigma_{k+1}'(h) = \sigma_k'(h) \cdot (I_k', \sigma_k')$, $k = 1, \ldots, m$. Since, $\sigma_1'(h) = \epsilon$, we thus derive by a straightforward induction that $\sigma_{k+1}'(h) = (I_1', \sigma_1') \cdots (I_k', \sigma_k')$, $k = 1, \ldots, m$. But $\sigma_{m+1}'(h) = \sigma_{l+1}(h)$, and consequently we derive that π' is a prefix of π. Since π is an extension of π' consisting of non-P_j steps only, by definition of $\mathcal{A}cz_W [\![P_j']\!]$ we subsequently derive that $\pi \in \mathcal{A}cz_W [\![P_j']\!]$. By proposition 20, $\mathcal{A}cz_V [\![P_1']\!] \cap \mathcal{A}cz_W [\![P_2']\!] \subseteq \mathcal{A}cz_{V \cup W} [\![P_1' \| P_2']\!]$. From this we derive that $\pi \in \mathcal{A}cz_{V \cup W} [\![P_1' \| P_2']\!]$. Since $\pi[\overline{V \cup W}] \models rely'$ and $\sigma_1 \models \phi'$ we thus infer from the validity of

$$\models_A \langle rely', guar \rangle : \{\phi'\} P_1' \| P_2' \{\psi\}$$

that $\langle \sigma, \sigma' \rangle \models guar$.

By an application of the parallel composition rule we thus obtain

$$\vdash \langle rely', guar \rangle : \{\phi'\} P_1' \parallel P_2' \{post_1 \wedge post_2\}.$$

In order to proceed we first show that $\models post_1 \wedge post_2 \rightarrow \psi$. Let $\sigma \models post_1 \wedge post_2$. By definition of $post_1$ and $post_2$ there exist computations

$$\pi = \langle \sigma_1, I_1, \sigma_2 \rangle \cdots \langle \sigma_n, I_n, \sigma_{n+1} \rangle \surd \in \mathcal{A}cz_{V_1} [\![P_i']\!]$$

and

$$\pi' = \langle \sigma_1', I_1', \sigma_2' \rangle \cdots \langle \sigma_m', I_m', \sigma_{m+1}' \rangle \surd \in \mathcal{A}cz_{V_2} [\![P_j']\!]$$

such that $\sigma = \sigma_{n+1} = \sigma_{m+1}'$, $\sigma_1 \models \phi'$, $\sigma_1' \models \phi'$, $\langle \sigma_k, \sigma_{k+1} \rangle \models rely_1$, $I_k \notin V_1$, and $\langle \sigma_k', \sigma_{k+1}' \rangle \models rely_2$, $I_k' \notin V_2$. By definition of $rely_i$ and construction of P_i' $(i = 1, 2)$, we may assume without loss of generality that $V_1 = \{P_1\}$, $V_2 = \{P_2\}$, $\sigma_{k+1}(h) = \sigma_k(h) \cdot (I_k, \sigma_k)$, $k = 1, \ldots, n$, and $\sigma_{k+1}'(h) = \sigma_k'(h) \cdot (I_k', \sigma_k')$, $k = 1, \ldots, m$ (simply rename the identifiers I_k, $k = 1, \ldots, n$, and I_l', $l = 1, \ldots, m$, accordingly). Since $\sigma_1(h) = \sigma_1'(h) = \epsilon$, we derive by a straightforward induction that $\sigma_{n+1}(h) = (I_1, \sigma_1) \cdots (I_n, \sigma_n)$ and $\sigma_{m+1}'(h) = (I_1', \sigma_1') \cdots (I_m', \sigma_m')$. Thus we derive from $\sigma_{n+1}(h) = \sigma_{m+1}'(h)$ that $\pi = \pi'$. Since $\mathcal{A}cz_{V_1} [\![P_1']\!] \cap \mathcal{A}cz_{V_2} [\![P_2']\!] \subseteq$

$\mathcal{A}cz_V [\![P_1' \| P_2']\!]$, we derive that $\pi \in \mathcal{A}cz_V [\![P_1' \| P_2']\!]$. By the given validity of the R-G-specification

$$\models_A \langle rely, guar \rangle : \{\phi\} P_1 \| P_2 \{\psi\}$$

(note that ϕ' implies ϕ and $rely'$ implies $rely$) we thus derive that $\sigma \models \psi$.

By an application of the consequence rule we thus obtain

$$\vdash \langle rely', guar \rangle : \{\phi'\} P_1' \parallel P_2' \{\psi\}.$$

Next we apply the auxiliary variables rule:

$$\vdash \langle \exists h.rely', guar \rangle : \{\exists h.\phi'\} P_1 \parallel P_2 \{\psi\}.$$

Finally, by an application of the consequence rule (using $\models rely \rightarrow \exists h.rely'$ and $\models \phi \rightarrow \exists h.\phi'$), we conclude

$$\vdash \langle rely, guar \rangle : \{\phi\} P_1 \parallel P_2 \{\psi\}.$$

9 Conclusion, Future, and Related Work

This paper advocates the usefulness of a semantic analysis of proof methods for concurrency. Such an analysis abstracts away from any expressibility issues and is especially effective in case of giving soundness and completeness proofs. By focussing on the semantic issues we discovered facts which were not known before about the R-G paradigm: that reactive-sequence semantics are inappropriate for modeling this paradigm, that Aczel-trace semantics does provide a correct interpretation for R-G validity, and that by adding finite stutter steps to reactive sequences a model is obtained which does model R-G validity adequately.

Furthermore, in such a semantic analysis one separates reasoning about sequential components from reasoning about parallel composition, by defining for the former an appropriate concept of inductive assertion networks (here: R-G-inductive assertion networks), and reasoning about the latter by Hoare-like proof rules. This considerably simplifies the reasoning process (just compare [15]), and focusses attention on the one central issue, namely, how to formulate a minimal number of rules for reasoning compositionally about shared-variable concurrency for open systems in a sound and semantically complete way. Such rules provide the basis for machine-supported compositional reasoning about concurrency in PVS, as used in, e.g., Hooman's work [8].

Finally, by focussing on the essential elements underlying completeness of the proposed proof method we discovered a proof which is much simpler than any previous "proof" appearing in the literature (of the correctness of none of which we are convinced anymore), and which extends the usual patterns of completeness proofs for Hoare-like reasoning about concurrency in a straightforward way.

This work arose out of a careful analysis of of the completeness proof presented in [14, 15], which is based on reduction to the completeness proof of the method of Owicki & Gries. We believe that our direct completeness proof

provides more insight in the R-G proof method. Also it is much simpler and therefore easier to check its correctness.

An interesting avenue of research opens up by applying the various methods which Gérard Berry employed, in his characterizations of the semantics of Esterel, to the Assume-Guarantee paradigm (the name of which was invented by Natarajan Shankar).

The present paper is the third one in a series of papers on the semantical analysis of compositional proof methods for concurrency, and will eventually appear as part of a chapter on compositional proof methods for concurrency in [6].

References

[1] M. Abadi and G. D. Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1):3–30, 1993.

[2] G. Berry. The Constructive Semantics of Esterel. Book in preparation, `http://www-sop.inria.fr/meije/esterel/doc/main-papers.html`, 1999.

[3] S. Brookes. *A fully abstract semantics of a shared variable parallel language.* In Proceedings 8th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, pages 98–109, 1993.

[4] F.S. de Boer, J.N. Kok, C. Palamedessi, and J.J.M.M. Rutten. The failure of failures: towards a paradigm for asynchronous communication. In Baeten and Groote, editors, *CONCUR'91*, LNCS 527. Springer-Verlag, 1991.

[5] W.-P. de Roever. The quest for compositionality - a survey of assertion-based proof systems for concurrent programs, part 1: Concurrency based on shared variables. In *Proc. of IFIP Working Conf, The Role of Abstract Models in Computer Science, North-Holland*, 1985.

[6] W.-P. de Roever, F.S. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. Concurrency Verification: An Introduction to State-based Methods. To appear.

[7] R.W. Floyd. Assigning meanings to programs. In *Proceedings AMS Symp. Applied Mathematics*, volume 19, pages 19–31, Providence, R.I., 1967. American Mathematical Society.

[8] J.Hooman. Compositional Verification of Real-Time Applications. In W.-P. de Roever, H. Langmaack, and A. Pnueli (eds.) *Compositionality: The Significant Difference. International Symposium, COMPOS'97, Bad Malente, Germany, September 8 –12, 1997.* pp. 130–149, Springer-Verlag, LNCS 1536, 1998.

[9] C.B. Jones. *Development methods for computer programs including a notion of interference.* PhD thesis, Oxford University Computing Laboratory, 1981.

[10] C.B. Jones. Tentative steps towards a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.

[11] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3), pp. 872–923, 1994.

[12] J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engeneering*, 7(7):417–426, 1981.

[13] E. Stark. A proof technique for rely/guarantee properties. In *Proceedings of 5th Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 206, pages 369–391. Springer-Verlag, 1985.

[14] Q. Xu. *A theory of state-based parallel programming.* DPhil. Thesis, Oxford University computing Laboratory, 1992.

[15] Q. Xu, W.-P. de Roever, and J. He. The rely-guarantee method for verifying shared-variable concurrent programs. *Formal Aspects of Computing*, 9(2):149–174, 1997.