On Rely-Guarantee Reasoning*

Stephan van Staden***

University College London
s.vanstaden@cs.ucl.ac.uk

Abstract. Many semantic models of rely-guarantee have been proposed in the literature. This paper proposes a new classification of the approaches into two groups based on their treatment of guarantee conditions. To allow a meaningful comparison, it constructs an abstract model for each group in a unified setting. The first model uses a weaker judgement and supports more general rules for atomic commands and disjunction. However, the stronger judgement of the second model permits the elegant separation of the rely from the guarantee due to Hayes et al. and allows refinement-style reasoning. The generalisation to models that use binary relations for postconditions is also investigated. An operational semantics is derived and both models are shown to be sound with respect to execution. All proofs have been checked with Isabelle/HOL and are available online.

Keywords: rely-guarantee, concurrency, semantics, soundness

1 Introduction

Rely-guarantee [8] is a well-established technique for reasoning about concurrent programs. It has been used to verify the correctness of tricky concurrent algorithms and inspired recent program logics such as RGSep, SAGL and LRG. It offers a compositional rule for concurrency by augmenting the usual pre- and postcondition specifications of Hoare logic with summaries of the interference of a program's concurrent environment and also of the program itself. The program can *rely* on its environment to behave according to the environment's interference specification, and must *guarantee* that it will adhere to its own interference constraints. Concretely, interference is summarised by a binary relation on states that over-approximates the effect of individual execution steps.

The judgements of rely-guarantee calculi are thus quintuples of the form:

$$Pre\, R\, \{Prog\}\, G\, Post$$

where Pre is the precondition (a set of states), R is the rely condition (a binary relation on states), $Prog$ is the program, G is the guarantee condition (a binary relation on states), and $Post$ is the postcondition (either a set of states or a binary relation on states, depending on the presentation).

Many semantic models of rely-guarantee have appeared in the literature. This paper proposes a new classification of the approaches into two broad groups, which essentially differ in their treatment of guarantee conditions. The main goals of the paper are

* This paper is dedicated to Ian Hayes.

*** Currently affiliated with Google Switzerland.

to capture these differences in a single abstract setting and to investigate their consequences.

The first group stipulates that each step of the program must satisfy the guarantee relation when the initial state satisfies the precondition and the environment satisfies the rely condition. Most mainstream models of rely-guarantee use this interpretation, for example [15,18,13,3,11,2].

The second group stipulates that each step of the program must satisfy the guarantee relation, irrespective of the initial state and the environment. Recent models of rely-guarantee that are based on refinement use this interpretation, for example [5,6,1]. The proofs in these papers suggest that the decoupling of the guarantee condition from the precondition and the rely enables refinement-style reasoning that is much more algebraic in flavour.

The models in previous work often differ in detail (e.g. programming constructs, operational semantics, etc.) which make it hard to study their merits and differences. To avoid this problem, the current paper uses a unified setting of traces as a semantic foundation. It then constructs two models of rely-guarantee that represent the two groups mentioned above.

In order to concentrate only on essential aspects, the semantic setting abstracts from many details. The presentation might therefore seem a bit unconventional to some readers. For example, it makes no assumptions about the (abstract) syntax of programs, it treats computational states abstractly, and it assumes no operational semantics. The idea is that such constraints can be added independently if and when needed. For example, we derive an operational calculus later in the paper to investigate whether the models are sound (i.e. correct) with respect to familiar small-step execution.

The judgements of the two models also make minimal demands. For example, the pre- and postconditions are not required to be 'stable' with respect to the rely condition, and the rely and/or guarantee relations need not be reflexive and/or transitive. Moreover, there is no fixed language for describing interference, and the same holds for assertions.

The models contribute several insights about the two semantic approaches:

- The judgement of the first model is weaker than the one of the second model. Despite this, the models validate mostly the same inference rules, but the weaker judgement supports more general rules for atomic commands and disjunction.
- Only the second model allows for the elegant decomposition of the rely-guarantee quintuple into $rely$ and $guar$ constructs due to Hayes et al. [5]. The separation of the rely from the guarantee permits refinement-style proofs which humans might find easier to construct.
- The models use postconditions that are single-state predicates, but they generalise nicely to models where postconditions are binary relations on states as is often the case in the rely-guarantee literature [8,2,5]. Interestingly, the attempt to generalise both models in a naive way fails because of their differences.
- Both models are sound with respect to big-step and small-step execution. The soundness proofs are not 'structural', but can be viewed as a simplification of the proof by Coleman and Jones [2]. The soundness results are decoupled from many operational concerns, such as the particular choice of execution rules, and the decision of which atomic operations are easy to implement in a computer.

This paper tries to present rely-guarantee incrementally from first principles, so it might be a good point to start learning about the main ideas. Working in a minimalistic setting also means that many of the proofs are shorter than their counterparts in other literature. All the proofs have been mechanised in Isabelle/HOL and are available online [7] to encourage further exploration.

Outline Section 2 describes the first model. Section 3 presents the second model which decouples the guarantee from the precondition and the rely. Section 4 generalises the models to rely-guarantee calculi where postconditions are binary relations and not sets of states. Section 5 derives operational calculi to show the soundness of the models with respect to execution. Section 6 discusses related work and Section 7 concludes.

2 The First Model

2.1 Formalising the Judgement

Many mainstream treatments of rely-guarantee (e.g. [18,13,11,17]) give the following informal meaning to the quintuple judgement $S\ R\ \{P\}\ G\ S'$:
if
1. program P is executed in a state which satisfies S, and (*precondition*)
2. every environment step satisfies R, (*rely*)

then
1. every step of P satisfies G, and (*guarantee*)
2. if the execution terminates, then the final state satisfies S'. (*postcondition*)

Here, and in the rest of the paper, P will be a program, S and S' are sets of states, and R and G are binary relations on states.

Instead of treating programs as syntactic objects that are generated by a particular (abstract) syntax, we model them generically as sets of traces. Each trace is a sequence of state pairs, called steps, that describe the program's ability to transform states. We use σ to range over states and t to range over traces. The empty trace is denoted by $[]$ and the infix operator : prepends a step to a trace.

Consider the trace $t = (\sigma_1, \sigma_1') : (\sigma_2, \sigma_2') : \cdots : (\sigma_n, \sigma_n')$. Step i transforms state σ_i into σ_i' before step $i+1$ can be executed. However, step $i+1$ does not have to be executed immediately after step i, because the model allows the (concurrent) environment to interfere between steps. If $\sigma_i' \neq \sigma_{i+1}$, for example, then step $i+1$ can only be executed when the environment interferes upon the completion of step i to transform state σ_i' into σ_{i+1}. A program's traces therefore describe its potential behaviour and allow for interference by concurrently executing programs. A typical program will have many traces that can never be observed in isolation. These "dormant" behaviours make concurrent programming especially tricky, and it is important to record them in the semantic model.

To formalise the informal interpretation of the rely-guarantee quintuple, it is helpful to consider how each trace of P must behave to satisfy the specification. This is the purpose of the auxiliary judgement $rg\text{-}trace$:

Definition 1. $\mathit{rg\text{-}trace}\ S\ R\ []\ G\ S' \stackrel{\mathit{def}}{=} R^*(S) \subseteq S'$
$\mathit{rg\text{-}trace}\ S\ R\ ((\sigma,\sigma'):t)\ G\ S' \stackrel{\mathit{def}}{=} \sigma \in R^*(S)\ \Rightarrow\ (\sigma,\sigma') \in G \wedge \mathit{rg\text{-}trace}\ \left\{\sigma'\right\}\ R\ t\ G\ S'$

The base case describes what should happen when the trace is empty. As the trace then contains no steps, it holds vacuously that every step is contained in G. Since the empty trace has no ability to alter the state, the environment must, irrespective of how many steps it performs, transform states satisfying the precondition into ones satisfying the postcondition. In formal terms, the image of S under the relation R^* must be contained in S', where R^* is the reflexive transitive closure of R.

The inductive case describes the situation for a non-empty trace whose first step is (σ,σ'). The first step can become enabled from precondition S and interference R whenever $\sigma \in R^*(S)$. If this is possible, then the step should be in G and the remainder of the trace must fulfill the specification where the new precondition $\left\{\sigma'\right\}$ is the result of the step.

The judgement for a program requires the corresponding auxiliary judgement for all its traces:

Definition 2. $S\ R\ \{P\}\ G\ S' \stackrel{\mathit{def}}{=} \forall t \in P : \mathit{rg\text{-}trace}\ S\ R\ t\ G\ S'$

2.2 Inference Rules

The definition allows a formal investigation of how the judgement can help us to reason about programs. Figures 1 and 2 show a collection of theorems in the form of inference rules[1]. (The reason for separating the rules in two figures is that only the ones of Figure 1 will also hold as theorems in the second model of Section 3.) Roughly speaking, there is one rule for reasoning about each programming operator, and there are additional rules for adapting the specification parts of a judgement.

The programming operators are the familiar ones from formal language theory and are summarised here for reference:

- *skip* is the language $\left\{[]\right\}$. It does nothing, because its only trace is empty and cannot transform the state.
- a stands for an atom, i.e. a program whose traces all have length one. Every such trace models a single step, so the atoms model (possibly nondeterministic) atomic operations. Atoms are isomorphic to binary relations on states. The binary relation that corresponds to atom a is given by $\mathit{rel}(a)$.
- $;$ is language concatenation. It corresponds to sequential composition of programs.
- $\|$ is language interleaving, also known as shuffle. It corresponds to concurrent composition.
- $\bigcup X$ is the union of all languages in X, i.e. the nondeterministic choice between programs in X. Its binary variant is $\cup$.
- * is the Kleene star, which iterates its operand zero or more times in sequence.
- *lfp* f is the least fixpoint of a monotone function f on languages. It is the meaning of a program P that is defined by recursion as $P = f(P)$. For example, P^* can be defined as the least fixpoint of the monotone function $(\lambda x\,.\,\mathit{skip} \cup (P\,;x))$.

(Jskip) $S\ R\ \{\mathit{skip}\}\ G\ (R^*(S))$

(Jseq) $S\ R\ \{P\}\ G\ S' \ \wedge\ S'\ R\ \{Q\}\ G\ S'' \ \Rightarrow\ S\ R\ \{P\,;Q\}\ G\ S''$

(Jconc) $S_1\ R_1\ \{P\}\ G_1\ S_1' \ \wedge\ S_2\ R_2\ \{Q\}\ G_2\ S_2' \ \wedge\ G_1 \subseteq R_2 \ \wedge\ G_2 \subseteq R_1 \ \Rightarrow$
$(S_1 \cap S_2)\ (R_1 \cap R_2)\ \{P \parallel Q\}\ (G_1 \cup G_2)\ (S_1' \cap S_2')$

(Jchoice) $(\forall P \in X : S\ R\ \{P\}\ G\ S') \ \Rightarrow\ S\ R\ \{\bigcup X\}\ G\ S'$

(Jiter) $S\ R\ \{P\}\ G\ S \ \wedge\ R(S) \subseteq S \ \Rightarrow\ S\ R\ \{P^*\}\ G\ S$

(Jrec) $(\forall P : S\ R\ \{P\}\ G\ S' \ \Rightarrow\ S\ R\ \{f(P)\}\ G\ S') \ \Rightarrow\ S\ R\ \{\mathit{lfp}\ f\}\ G\ S'$

(Jweak) $S_1\ R_1\ \{P\}\ G_1\ S_1' \ \wedge\ S_2 \subseteq S_1 \ \wedge\ R_2 \subseteq R_1 \ \wedge\ G_1 \subseteq G_2 \ \wedge\ S_1' \subseteq S_2' \ \Rightarrow$
$S_2\ R_2\ \{P\}\ G_2\ S_2'$

(Jstren) $S\ R\ \{P\}\ G\ S' \ \Rightarrow\ (R^*(S))\ R^*\ \{P\}\ (G \cap \mathit{steps}(P))\ (S' \cap (R \cup G)^*(S))$

(Jconj) $S_1\ R_1\ \{P\}\ G_1\ S_1' \ \wedge\ S_2\ R_2\ \{Q\}\ G_2\ S_2' \ \Rightarrow$
$(S_1 \cap S_2)\ (R_1 \cap R_2)\ \{P \cap Q\}\ (G_1 \cap G_2)\ (S_1' \cap S_2')$

Fig. 1. Common rely-guarantee inference rules

Most of the rules in Figure 1 are self-explanatory, so a few observations should be sufficient to see how they operate:

- (Jskip): Any guarantee, including the empty relation, is acceptable because *skip* performs no steps. The postcondition takes into account that the environment might still transform states that satisfy the precondition.
- (Jseq): The postcondition of the first program must be the precondition of the second one.
- (Jconc): The guarantee of each program must be compatible with what its concurrent partner relies on. In the consequent of the rule, the environment can only do what both components have relied on, while the guarantee condition accommodates steps of both components.
- (Jchoice): When every program in a collection satisfies a specification, then the nondeterministic choice between them will also satisfy it.
- (Jiter): Interference from the environment should not invalidate S, which functions as the loop invariant.
- (Jrec): To verify a recursive program, it suffices to check that unfolding the definition once meets the specification when all recursive occurrences meet it.
- (Jweak): This rule can weaken the specification of a judgement and is sometimes known as the 'rule of consequence'.
- (Jstren): This rule can stengthen a specification, i.e. enlarge the precondition and rely, and shrink the guarantee and postcondition. This is not magic – it simply exploits redundancy that exists in specifications. It is simplest to understand this rule in a piecewise fashion as follows. If a judgement holds with precondition S and rely condition R, then the precondition $R^*(S)$ will also work. Moreover, the environment can safely do steps described by R^*. The program can at most perform the steps mentioned in its traces, so the guarantee condition can always be restricted

[1] The prefix 'J' in the names of inference rules stands for 'Jones' in tribute to [8].

to them ($steps(P)$ relates a pair of states iff the pair is a step in some trace of P). And finally, since the rely and guarantee conditions over-approximate what the environment and program can respectively do, their combination cannot yield final states that are outside $(R \cup G)^*(S)$.
– (Jconj): This rule conjoins the specifications of two judgements. It allows the programs to be different and intersects them in the resulting judgement.

(J1atom) $\quad rel(a) \cap (R^*(S))\times\Sigma \subseteq G \ \wedge \ (R^*; rel(a); R^*)(S) \subseteq S' \ \Rightarrow \ S\ R\ \{a\}\ G\ S'$

(J1disj) $\quad (\forall S \in Y : S\ R\ \{P\}\ G\ S') \ \Rightarrow \ (\bigcup Y)\ R\ \{P\}\ G\ S'$

Fig. 2. Rules that are specific to the first model

The rules in Figure 2 should not present difficulties either:

– (J1atom) requires the guarantee to include the relation on states that is isomorphic to the atom, but allows restricting the domain of this relation to those states that the environment can reach from the precondition in zero or more steps. (Σ is the set of all states and $\times$ the Cartesian product operator.) The restriction captures the fact that the atom will never have to perform its step from any other state. Furthermore, the postcondition must at least include those states that can be reached from the precondition under $R^*; rel(a); R^*$, which describes the effect of interleaving the atomic operation with environment steps from R. (When applied to binary relations, ; is the familiar composition operator. Another common symbol for it is $\circ$.)
– (J1disj) says that if a program can meet a specification from each precondition in a set, then it must also be able to meet the specification from their union.

As in most calculi, there is some degree of flexibility in the presentation of the rules. For example, instead of (Jskip) one can prefer the rule:

$$R^*(S) \subseteq S' \ \Rightarrow \ S\ R\ \{\mathit{skip}\}\ G\ S'$$

and it is possible to collapse (J1atom) to:

$$S\ R\ \{a\}\ (rel(a) \cap (R^*(S))\times\Sigma)\ ((R^*; rel(a); R^*)(S))$$

In both cases one can justify the change and regain the original rule with the weakening rule (Jweak), so nothing is really gained or lost. It is also easy to build aspects of the strengthening rule (Jstren) into other rules, because (Jstren) and (Jweak) imply equivalences such as:

$$\begin{aligned} S\ R\ \{P\}\ G\ S' \ &\Leftrightarrow \ (R^*(S))\ R\ \{P\}\ G\ S' \\ S\ R\ \{P\}\ G\ S' \ &\Leftrightarrow \ S\ R^*\ \{P\}\ G\ S' \\ S\ R\ \{P\}\ G\ S' \ &\Leftrightarrow \ S\ R\ \{P\}\ G\ (S' \cap (R \cup G)^*(S)) \end{aligned}$$

These equivalences will of course hold in *any* model that validates (Jstren) and (Jweak).

2.3 Proofs

The formal justification of the inference rules requires rigorous proofs. There are two main reasons why such proofs can be instructive. Firstly, they describe in detail why each rule must work. Secondly, the proofs can collectively communicate the general style of reasoning that the model promotes. It will become clear in Section 3 that both aspects can differ considerably between models.

The rely-guarantee judgement in the current model is defined in terms of the auxiliary judgement $rg\text{-}trace$. Many rules will therefore directly follow from similar ones about $rg\text{-}trace$. Induction on traces is the main mathematical tool of this model, as $rg\text{-}trace$ was defined by recursion on traces.

For example, the rule (Jconc) follows directly from the following lemma about $rg\text{-}trace$, where $t_1 \otimes t_2$ denotes the set of all interleavings of traces t_1 and t_2. Unsurprisingly, the proof proceeds by induction on t:

Lemma 1. $rg\text{-}trace\ S_1\ R_1\ t_1\ G_1\ S_1' \ \wedge \ rg\text{-}trace\ S_2\ R_2\ t_2\ G_2\ S_2' \ \wedge \ G_1 \subseteq R_2 \ \wedge \ G_2 \subseteq R_1 \ \wedge \ t \in t_1 \otimes t_2 \ \Rightarrow \ rg\text{-}trace\ (S_1 \cap S_2)\ (R_1 \cap R_2)\ t\ (G_1 \cup G_2)\ (S_1' \cap S_2')$

Proof. By induction on the structure of t:

- *Base case. We must show that it holds for $t = []$. Since $t \in t_1 \otimes t_2$, we know $t_1 = []$ and $t_2 = []$. Expanding the two $rg\text{-}trace$ assumptions gives ${R_1}^*(S_1) \subseteq S_1'$ and ${R_2}^*(S_2) \subseteq S_2'$, which in turn imply $(R_1 \cap R_2)^*(S_1 \cap S_2) \subseteq S_1' \cap S_2'$ because the Kleene star and relational image operators are both monotone.*
- *Step case. Suppose the property holds for a trace t for all t_1, t_2, S_1, S_2. We must show that it will also hold for $(\sigma, \sigma') : t$. So assume the rule's antecedents $rg\text{-}trace\ S_1\ R_1\ t_1\ G_1\ S_1'$ and $rg\text{-}trace\ S_2\ R_2\ t_2\ G_2\ S_2'$ and $G_1 \subseteq R_2$ and $G_2 \subseteq R_1$ and $(\sigma, \sigma') : t \in t_1 \otimes t_2$. The last assumption implies $\exists t_1' : t_1 = (\sigma, \sigma') : t_1' \ \wedge \ t \in t_1' \otimes t_2$ or $\exists t_2' : t_2 = (\sigma, \sigma') : t_2' \ \wedge \ t \in t_1 \otimes t_2'$. The two cases are symmetric, so we will only show the reasoning for the first one. The goal is to show that $\sigma \in (R_1 \cap R_2)^*(S_1 \cap S_2)$ implies both $(\sigma, \sigma') \in G_1 \cup G_2$ and $rg\text{-}trace \left\{\sigma'\right\} (R_1 \cap R_2)\ t\ (G_1 \cup G_2)\ (S_1' \cap S_2')$.*

  Suppose $\sigma \in (R_1 \cap R_2)^*(S_1 \cap S_2)$. Then $\sigma \in {R_1}^*(S_1)$ and $\sigma \in {R_2}^*(S_2)$. Expanding the $rg\text{-}trace$ assumption for t_1 now gives $rg\text{-}trace \left\{\sigma'\right\} R_1\ t_1'\ G_1\ S_1'$ and $(\sigma, \sigma') \in G_1$. So clearly $(\sigma, \sigma') \in G_1 \cup G_2$. Moreover, $(\sigma, \sigma') \in G_1$ and $G_1 \subseteq R_2$ and $\sigma \in {R_2}^*(S_2)$ imply that $\sigma' \in {R_2}^*(S_2)$. So we can enlarge the precondition in the $rg\text{-}trace$ assumption for t_2 to ${R_2}^*(S_2)$ and then shrink the precondition of the result to $\left\{\sigma'\right\}$. Applying the induction hypothesis to $rg\text{-}trace \left\{\sigma'\right\} R_1\ t_1'\ G_1\ S_1'$ and $rg\text{-}trace \left\{\sigma'\right\} R_2\ t_2\ G_2\ S_2'$ concludes the proof.

This proof and proofs for all the other rules have been mechanised in Isabelle/HOL. The proof script is available online [7]. The level of detail in formal proofs is typically greater than in pen-and-paper ones, so they can help to clarify gaps in the reasoning. They can also help to formulate different proofs of existing results, or to explore variations on the rules showed here, or even to establish the validity of entirely new ones.

Another use of the Isabelle/HOL formalisation is the discovery of counterexamples. It is straightforward to show, for example, that neither the precondition nor the postcondition have to be 'stable' with respect to the rely condition in this model. In other words, there are examples where $S\ R\ \{P\}\ G\ S'$ holds, yet $R(S) \not\subseteq S$ and $R(S') \not\subseteq S'$. Similarly, the guarantee condition need not be reflexive. Section 6 will discuss why such constraints can be useful in more concrete settings, but there was no need to impose them here.

2.4 The Bigger Picture

The introduction mentioned that rely-guarantee is a generalisation of Hoare logic that augments judgements with rely and guarantee conditions. This intuition can be formalised in a theorem that relates the rely-guarantee quintuple to the Hoare triple (an intuitive understanding of the Hoare triple suffices here; its formal treatment is postponed to Section 5):

Theorem 1. $(\exists R\ G : S\ R\ \{P\}\ G\ S') \ \Leftrightarrow\ S\,\{P\}\,S'$

The theorem says that one can establish the Hoare triple $S\,\{P\}\,S'$ by finding some rely R and guarantee G and establishing the rely-guarantee judgement $S\ R\ \{P\}\ G\ S'$ instead. Moreover, if the Hoare triple holds, then it will always be possible to find appropriate rely and guarantee conditions.

One way to prove the theorem is to show that the Hoare triple corresponds exactly to the 'interference-free' situation where the environment can do nothing (the rely is empty) but the program can do anything (the guarantee is the universal relation on states):

Lemma 2. $S\ \emptyset\ \{P\}\ (\Sigma{\times}\Sigma)\ S' \ \Leftrightarrow\ S\,\{P\}\,S'$

Theorem 1 then follows immediately by (Jweak).

Lemma 2 says that the Hoare triple is a special case of the rely-guarantee judgement. Alternatively, one can view it as characterising certain rely-guarantee judgements (those with empty rely and universal guarantee conditions) in terms of Hoare logic. Surprisingly, the next result shows that it is possible to extend this characterisation to account for arbitrary rely conditions:

Lemma 3. $S\ R\ \{P\}\ (\Sigma{\times}\Sigma)\ S' \ \Leftrightarrow\ S\,\{P \parallel \mathit{traces}(R)\}\,S'$

In formal language terminology, $\mathit{traces}(R)$ is the set of all words over alphabet R. Now $\mathit{traces}(\emptyset) = \mathit{skip}$, and *skip* is the unit of $\parallel$, so Lemma 2 is a straightforward consequence.

One can prove Lemma 3 by first showing:

$$\mathit{rg\text{-}trace}\ S\ R\ t\ (\Sigma{\times}\Sigma)\ S' \ \Leftrightarrow\ S\,\{\{\!|t|\!\} \parallel \mathit{traces}(R)\}\,S'$$

This lemma holds by induction on t and [7] contains the full proof.

Unfortunately, there appears to be no straightforward way to extend the characterisation of Lemma 3 to cover arbitrary guarantee conditions. A guarantee condition in this

model is quite complicated: its fulfillment depends not only on the program, but also on the precondition and the behaviour of the environment. This is also the case in most mainstream treatments of rely-guarantee. In [2], for example, the auxiliary judgement $\{S, R\} \models P$ **within** G makes the dependency very clear.

3 The Second Model

The quintuple of the second model combines the insight of Lemma 3 with a simple treatment of guarantee conditions:

Definition 3. $S\ R\ \{P\}\ G\ S' \stackrel{def}{=} S\ \{P \parallel traces(R)\}\ S' \ \wedge\ P \subseteq traces(G)$

It demands that all the steps[2] of P must be in G, irrespective of the precondition and the rely. The informal meaning of the quintuple is therefore:
every step of program P satisfies G, and (*guarantee*)
if
1. P is executed in a state which satisfies S, and (*precondition*)
2. every environment step satisfies R, (*rely*)

then
1. if the execution terminates, then the final state satisfies S'. (*postcondition*)

It is easy to show that the new judgement is stronger than the previous one.

3.1 Inference Rules

The fact that the judgement is stronger means that we must again determine which inference rules are theorems. Fortunately, it turns out that all the rules in Figure 1 remain valid in this model. The ones in Figure 2 are now invalid, but one can use the variants that appear in Figure 3:

(J2atom) $\quad rel(a) \subseteq G \ \wedge\ (R^*; rel(a); R^*)(S) \subseteq S' \ \Rightarrow\ S\ R\ \{a\}\ G\ S'$

(J2disj) $\quad Y \neq \emptyset \ \wedge\ (\forall S \in Y : S\ R\ \{P\}\ G\ S') \ \Rightarrow\ (\bigcup Y)\ R\ \{P\}\ G\ S'$

Fig. 3. Rules that are specific to the second model

- (J2atom): Note that $rel(a)$ must now be fully included in G – there is no possibility to restrict its domain to $R^*(S)$ before checking the inclusion.
- (J2disj): The additional restriction that Y should not be empty reflects the fact that $\emptyset\ R\ \{P\}\ G\ S'$ is not a theorem in this model: the empty precondition does *not* ensure that the steps of P are included in G.

The additional restriction in the rule of disjunction might be a cause for concern. However, by and large the judgement is well-behaved. The next two subsections show that it supports elegant proofs and an interesting decomposition of the judgement.

[2] Note the Galois connection $steps(P) \subseteq G \ \Leftrightarrow\ P \subseteq traces(G)$.

3.2 Proofs

The seemingly minor act of decoupling the satisfaction of the guarantee condition from the precondition and the rely has a significant impact on the style of the proofs. It becomes possible to justify the inference rules with algebraic reasoning that leverages program refinement. This is similar in spirit to more recent formulations of rely-guarantee [5,6,1].

In our simple setting, we say that P refines P' if and only if $P \subseteq P'$. Here is a refinement-style proof of (Jconc), for example:

Lemma 4. $S_1\, R_1\, \{P\}\, G_1\, S_1' \;\wedge\; S_2\, R_2\, \{Q\}\, G_2\, S_2' \;\wedge\; G_1 \subseteq R_2 \;\wedge\; G_2 \subseteq R_1 \;\Rightarrow\; (S_1 \cap S_2)\, (R_1 \cap R_2)\, \{P \parallel Q\}\, (G_1 \cup G_2)\, (S_1' \cap S_2')$

Proof. Assume the antecedents. The first one gives $S_1\, \{P \parallel \mathit{traces}(R_1)\}\, S_1'$ *and* $P \subseteq \mathit{traces}(G_1)$. *The second gives* $S_2\, \{Q \parallel \mathit{traces}(R_2)\}\, S_2'$ *and* $Q \subseteq \mathit{traces}(G_2)$. *Clearly* $P \parallel Q \subseteq \mathit{traces}(G_1 \cup G_2)$, *so it remains to show the validity of the Hoare triple* $S_1 \cap S_2\, \{P \parallel Q \parallel \mathit{traces}(R_1 \cap R_2)\}\, S_1' \cap S_2'$. *Consider the refinement development:*

$$
\begin{array}{ll}
 & P \parallel \mathit{traces}(R_1) \\
= & \textit{// Since } \mathit{traces}(R_1) = \mathit{traces}(R_1) \parallel \mathit{traces}(R_1). \\
 & P \parallel \mathit{traces}(R_1) \parallel \mathit{traces}(R_1) \\
\supseteq & \textit{// Because } G_2 \subseteq R_1. \\
 & P \parallel \mathit{traces}(G_2) \parallel \mathit{traces}(R_1) \\
\supseteq & \textit{// By assumption, } Q \subseteq \mathit{traces}(G_2). \\
 & P \parallel Q \parallel \mathit{traces}(R_1) \\
\supseteq & \textit{// By simple monotonicity.} \\
 & P \parallel Q \parallel \mathit{traces}(R_1 \cap R_2)
\end{array}
$$

Since Hoare triples remain valid for refined programs, $S_1\, \{P \parallel Q \parallel \mathit{traces}(R_1 \cap R_2)\}\, S_1'$ *holds. By a symmetric argument, we obtain* $S_2\, \{P \parallel Q \parallel \mathit{traces}(R_1 \cap R_2)\}\, S_2'$. *Applying the Hoare rule of conjunction to these triples completes the proof.*

3.3 Decomposition of the Quintuple

Another interesting consequence of the judgement's definition is that it can be decomposed into 'rely' and 'guar' constructs along the lines of [5].

Let $P \mathbin{-\!\!\parallel} P' \stackrel{\text{def}}{=} \bigcup \left\{ P'' \mid P'' \parallel P \subseteq P' \right\}$. Consider the following definitions:

Definition 4. $\mathit{rely}\ R\ P \stackrel{\mathit{def}}{=} \mathit{traces}(R) \mathbin{-\!\!\parallel} P$

Definition 5. $\mathit{guar}\ G\ P \stackrel{\mathit{def}}{=} \mathit{traces}(G) \cap P$

The Galois connection $P \parallel P' \subseteq P'' \;\Leftrightarrow\; P \subseteq P' \mathbin{-\!\!\parallel} P''$ and Definition 4 imply that $\mathit{rely}\ R\ P$ is the largest program (i.e. the least refined or the most nondeterministic one) that, when placed in an environment R, will refine P:

Lemma 5. $P' \parallel \mathit{traces}(R) \subseteq P \;\Leftrightarrow\; P' \subseteq \mathit{rely}\ R\ P$

Definition 5 is simpler: $guar\ G\ P$ is the largest program that refines P whose steps are all in G.

Let $[S, S']$ denote the 'specification statement' [10], i.e. the largest program that satisfies the Hoare triple with precondition S and postcondition S'. Formally, $[S, S']$ can be defined as $\bigcup \left\{ P \mid S\,\{P\}\,S' \right\}$. Then $S\,\{P\}\,S' \;\Leftrightarrow\; P \subseteq [S, S']$. Lemma 5 implies $S\,\{P \parallel traces(R)\}\,S' \;\Leftrightarrow\; P \subseteq rely\ R\ [S, S']$, so the $rely$ and $guar$ constructs elegantly factor the judgement into smaller parts:

Lemma 6. $S\ R\ \{P\}\ G\ S' \;\Leftrightarrow\; P \subseteq guar\ G\ (rely\ R\ [S, S'])$

Instead of using the inference rules in Figures 1 and 3, one can use refinement and the algebraic properties of $rely$ and $guar$ to the same effect. This alternative way to reason about programs allows for a more general presentation, as there is no obligation to restrict attention to constructs of the form $guar\ G\ (rely\ R\ [S, S'])$. The work by Hayes et al. [5] offers an excellent example of this approach.

3.4 Bigger Picture

It is not hard to see that Theorem 1 and lemmas 2 and 3 remain valid in this model. So once again the quintuple represents a conservative extension of the Hoare triple. However, the two models provide different extensions! This raises the exciting possibility that new extensions with pleasant properties might still await future discovery.

From a practical point of view, one might prefer the rules of the first model, because (J1atom) and (J1disj) are more powerful than (J2atom) and (J2disj). The second model will generally use larger guarantee conditions, so in order to apply the concurrency rule (Jconc), the rely conditions must also be larger. Whether this will create problems during the verification of concrete programs remains to be seen. The examples in [5], which uses $rely$ and $guar$ constructs, and [1], which uses a quintuple judgement, suggest that this is perhaps not a serious drawback.

4 Using Binary Relations for Postconditions

Many treatments of rely-guarantee (e.g. [8,2,5]) do not use postconditions that are sets of states. Instead, they use predicates that relate the pre and the post state, i.e. binary relations on states. Some treatments of Hoare logic also follow this convention. The Hoare triple with a relation T in the postcondition can be defined in terms of the usual one as follows[3]:

Definition 6. $S\,\{P\}\,T \;\stackrel{def}{=}\; \forall \sigma \in S : \left\{\sigma\right\}\,\{P\}\,T(\left\{\sigma\right\})$

This suggests a similar definition for the rely-guarantee quintuple where postconditions are relations:

Definition 7. $S\ R\ \{P\}\ G\ T \;\stackrel{def}{=}\; \forall \sigma \in S : \left\{\sigma\right\}\ R\ \{P\}\ G\ (T(\left\{\sigma\right\}))$

(Rskip) $S\,R\,\{\mathit{skip}\}\,G\,R^*$

(Rseq) $S\,R\,\{P\}\,G\,(T \cap \Sigma{\times}S')\ \wedge\ S'\,R\,\{Q\}\,G\,T'\ \Rightarrow\ S\,R\,\{P\,;Q\}\,G\,(T;T')$

(Rconc) $S\,(R{\cup}G_2)\,\{P\}\,G_1\,T_1\ \wedge\ S\,(R{\cup}G_1)\,\{Q\}\,G_2\,T_2\ \Rightarrow$
$S\,R\,\{P \parallel Q\}\,(G_1{\cup}G_2)\,(T_1{\cap}T_2{\cap}(R{\cup}G_1{\cup}G_2)^*)$

(Rchoice) $(\forall P \in X : S\,R\,\{P\}\,G\,T)\ \Rightarrow\ S\,R\,\{\bigcup X\}\,G\,T$

(Riter) $S\,R\,\{P\}\,G\,(T' \cap \Sigma{\times}S)\ \wedge\ R^* \cap S{\times}\Sigma \subseteq T\ \wedge\ T';T \subseteq T\ \Rightarrow\ S\,R\,\{P^*\}\,G\,T$

(Rrec) $(\forall P : S\,R\,\{P\}\,G\,T\ \Rightarrow\ S\,R\,\{f(P)\}\,G\,T)\ \Rightarrow\ S\,R\,\{\mathit{lfp}\,f\}\,G\,T$

(Rweak) $S_1\,R_1\,\{P\}\,G_1\,T_1\ \wedge\ S_2 \subseteq S_1\ \wedge\ R_2 \subseteq R_1\ \wedge\ G_1 \subseteq G_2\ \wedge\ T_1 \subseteq T_2\ \Rightarrow$
$S_2\,R_2\,\{P\}\,G_2\,T_2$

(Rstren) $S\,R\,\{P\}\,G\,T\ \Rightarrow\ S\,R^*\,\{P\}\,(G{\cap}\mathit{steps}(P))\,(T \cap (R{\cup}G)^* \cap S{\times}\Sigma)$

(Rconj) $S_1\,R_1\,\{P\}\,G_1\,T_1\ \wedge\ S_2\,R_2\,\{Q\}\,G_2\,T_2\ \Rightarrow$
$(S_1{\cap}S_2)\,(R_1{\cap}R_2)\,\{P \cap Q\}\,(G_1{\cap}G_2)\,(T_1{\cap}T_2)$

Fig. 4. Rely-guarantee rules with relations for postconditions

It is now possible to explore the conditions under which this definition validates familiar inference rules. For example, the rule[4] (Rweak) in Figure 4 follows directly from (Jweak). Similarly, (Rconc) holds by (Jconc), (Jweak) and (Jstren). One can also show that (Rseq) follows from (Jseq), (Jweak) and (J1disj). But (J1disj) is not valid in the second model! Indeed, (Rseq) is not a theorem when Definition 7 is applied to its quintuple. So although the naive approach of Definition 7 successfully generalises the first model to the setting where postconditions are relations, it fails to generalise the second model whose quintuple behaves slightly differently.

Nevertheless, it is possible to mirror Definition 3 using the Hoare triple of Definition 6:

Definition 8. $S\,R\,\{P\}\,G\,T\ \stackrel{\mathit{def}}{=}\ S\,\{P \parallel \mathit{traces}(R)\}\,T\ \wedge\ P \subseteq \mathit{traces}(G)$

This definition facilitates algebraic proofs of the inference rules in Figure 4, so it successfully generalises the second model. It is also straightforward to see that the rely and guar constructs of Section 3.3 need no adaptation: if $[S,T]$ denotes the specification statement with relation T as postcondition, i.e. $[S,T]\ \stackrel{\text{def}}{=}\ \bigcup\left\{P \mid S\,\{P\}\,T\right\}$, then the judgement of Definition 8 satisfies $S\,R\,\{P\}\,G\,T\ \Leftrightarrow\ P \subseteq \mathit{guar}\,G\,(\mathit{rely}\,R\,[S,T])$.

Definition 8 validates the following equivalence where the second model's quintuple appears in the right-hand side:

$$S\,R\,\{P\}\,G\,T\ \Leftrightarrow\ (\forall\sigma \in S : \left\{\sigma\right\}\,R\,\{P\}\,G\,(T(\left\{\sigma\right\})))\ \wedge\ \emptyset\,R\,\{P\}\,G\,\emptyset$$

This equivalence shows that Definition 8 strengthens Definition 7 with an additional conjunct that caters specifically for the case where the precondition is false.

[3] Because of the equivalence $S\,\{P\}\,S'\ \Leftrightarrow\ S\,\{P\}\,S{\times}S'$, it is also possible to go in the opposite direction, i.e., one can define the usual triple in terms of the triple where postconditions are relations.

[4] The prefix 'R' in the names of inference rules stands for 'relation'.

The apparent differences between Definitions 7 and 8 can now be resolved by noticing their underlying unity – the generalised judgements of both models satisfy:

Lemma 7. $S\,R\,\{P\}\,G\,T \;\Leftrightarrow\; \forall S' \subseteq S : S'\,R\,\{P\}\,G\,(T(S'))$

Of course this does not imply that the two generalised models will support the same rules. Figure 4 contains some common rules[5], while Figures 5 and 6 show rules that are specific to each generalisation. Notice that Figures 5 and 6 mirror the differences that were present in Figures 2 and 3.

(R1atom) $\quad rel(a) \cap (R^*(S))\times\Sigma \subseteq G \;\wedge\; (R^*; rel(a); R^*) \subseteq T \;\Rightarrow\; S\,R\,\{a\}\,G\,T$

(R1disj) $\quad (\forall S \in Y : S\,R\,\{P\}\,G\,T) \;\Rightarrow\; (\bigcup Y)\,R\,\{P\}\,G\,T$

Fig. 5. Rules specific to the generalisation of the first model

(R2atom) $\quad rel(a) \subseteq G \;\wedge\; (R^*; rel(a); R^*) \subseteq T \;\Rightarrow\; S\,R\,\{a\}\,G\,T$

(R2disj) $\quad Y \neq \emptyset \;\wedge\; (\forall S \in Y : S\,R\,\{P\}\,G\,T) \;\Rightarrow\; (\bigcup Y)\,R\,\{P\}\,G\,T$

Fig. 6. Rules specific to the generalisation of the second model

In retrospect, one can see that since $S\,\{P\}\,T \;\Leftrightarrow\; \forall S' \subseteq S : S'\,\{P\}\,T(S')$ is also valid, it would have been easier to start the generalisation from this characterisation instead of the one in Definition 6. In general, defining judgements in terms of others is a powerful technique to construct sophisticated notions and inference rules in a stepwise fashion, but a little experimentation is often necessary to find suitable definitions for derived judgements.

5 Soundness

This paper presents the two models of rely-guarantee independently of operational calculi. The presentation is fairly self-contained – only the reference to Hoare logic involves another judgement. In fact the Hoare triple also has a direct definition that does not presuppose an operational judgement or calculus:

Definition 9. $S\,\{P\}\,S' \stackrel{def}{=}$
$IF\text{-}traces\text{-}ending\text{-}in(S)\,;P \subseteq IF\text{-}traces\text{-}ending\text{-}in(S') \cup WithInterference$

[5] When postconditions are sets of states, one can always change a precondition S into $R^*(S)$ where R is the rely condition. This is invalid when postconditions are relations between input and output states, so (Rstren) does not change the precondition.

Here, $IF\text{-}traces\text{-}ending\text{-}in(S)$ denotes the set of all traces that are interference-free and end in a state that is also in S. A trace is interference-free when, for each step in the trace, the second state of the step is the same as the first state of the next step if such a step exists. The set of all traces that are not interference-free is denoted by $WithInterference$. Definition 9 says that all the interference-free traces of P that start in a state in S must end in a state in S'. Moreover, if P contains the empty trace (which is trivially interference-free), then it must be the case that $S \subseteq S'$.

Despite the independence from operational calculi, the expected soundness relationships nonetheless hold. To show this, we first give direct definitions of familiar operational judgements and then prove that the soundness relationships are theorems. The same technique was used in [14] to demonstrate the soundness of the Views program logic.

The big-step operational judgement is defined as follows:

Definition 10. $\langle P, \sigma\rangle \longrightarrow \sigma' \stackrel{def}{=}$
$\exists t \in IF\text{-}traces\text{-}ending\text{-}in(\sigma) : \exists t' \in IF\text{-}traces\text{-}ending\text{-}in(\sigma') : \left\{t\right\}; P \supseteq \left\{t'\right\}$

It says that P has an interference-free trace that can transform the initial state σ into the final state σ'. The familiar soundness relationship holds between the Hoare triple and the big-step judgement ([7] contains a short and simple formal proof):

Lemma 8. $S\,\{P\}\,S' \;\Leftrightarrow\; (\forall \sigma \in S : \forall \sigma' : \langle P, \sigma\rangle \longrightarrow \sigma' \;\Rightarrow\; \sigma' \in S')$

This result and Theorem 1 imply that both models of rely-guarantee are sound with respect to big-step rules:

Theorem 2. $(\exists R\, G : S\, R\, \{P\}\, G\, S') \;\Leftrightarrow$
$(\forall \sigma \in S : \forall \sigma' : \langle P, \sigma\rangle \longrightarrow \sigma' \;\Rightarrow\; \sigma' \in S')$

However, it is much more common to establish the soundness of rely-guarantee with respect to a small-step judgement in the style of Plotkin [12]. The reason is that operational judgements are conventionally *defined* in terms of syntax-directed rules, and the fine-grained interleaving of concurrent composition cannot be expressed by considering only the big steps of each operand.

Although such considerations are not problematic in this more semantic treatment where judgements are not defined by sets of inference rules, it is quite easy to accommodate small-step calculi. The small-step judgement can be defined in terms of a set $Actions$ that contains the 'small' operations or actions that are easy to implement in a computer:

Definition 11. $\langle P, \sigma\rangle \longrightarrow \langle P', \sigma'\rangle \stackrel{def}{=} \exists Q \in Actions : P \supseteq Q; P' \wedge \langle Q, \sigma\rangle \longrightarrow \sigma'$

It says that one way of executing P is to execute some action followed by P'. The action itself is hidden – only its effect on the state is explicit in the judgement.

There is a simple relationship between the reflexive transitive closure of the small-step judgement and the big-step one:

Lemma 9. $\langle P, \sigma\rangle \longrightarrow^* \langle \mathit{skip}, \sigma'\rangle \;\Rightarrow\; \langle P, \sigma\rangle \longrightarrow \sigma'$

Whether or not the converse holds depends on the choice of $Actions$, but Lemma 9 is sufficient to prove the soundness of both rely-guarantee models with respect to small-step execution:

Theorem 3. $(\exists R\ G : S\ R\ \{P\}\ G\ S') \Rightarrow$
$(\forall \sigma \in S : \langle P, \sigma\rangle \longrightarrow^* \langle \mathit{skip}, \sigma'\rangle \Rightarrow \sigma' \in S')$

It is remarkable that this result is independent of the choice of machine-executable actions. The soundness also remains valid regardless of the choice of operational rules: any rule that is a theorem can be used to discover executions. For example, when $Actions$ includes *skip*, then the familiar operational rules in Figure 7 are all acceptable[6], and one can also (or alternatively) adopt the following rule for nondeterministic choice:

$$P \in X \ \wedge\ \langle P, \sigma\rangle \longrightarrow \langle P', \sigma'\rangle \ \Rightarrow\ \langle \bigcup X, \sigma\rangle \longrightarrow \langle P', \sigma'\rangle$$

None of these decisions or changes can jeopardise the validity of Theorem 3. The formalisation effectively decouples deductive (i.e. program logic) and operational concerns, yet it enforces soundness at the same time.

(Patom)	$a \in \mathit{Actions} \ \wedge\ (\sigma,\sigma') \in \mathit{rel}(a) \ \Rightarrow\ \langle a, \sigma\rangle \longrightarrow \langle \mathit{skip}, \sigma'\rangle$
(Pseq1)	$\langle \mathit{skip}\,; P, \sigma\rangle \longrightarrow \langle P, \sigma\rangle$
(Pseq2)	$\langle P, \sigma\rangle \longrightarrow \langle P', \sigma'\rangle \ \Rightarrow\ \langle P\,; P'', \sigma\rangle \longrightarrow \langle P'\,; P'', \sigma'\rangle$
(Pchoice)	$P \in X \ \Rightarrow\ \langle \bigcup X, \sigma\rangle \longrightarrow \langle P, \sigma\rangle$
(Piter1)	$\langle P^*, \sigma\rangle \longrightarrow \langle \mathit{skip}, \sigma\rangle$
(Piter2)	$\langle P^*, \sigma\rangle \longrightarrow \langle P\,; P^*, \sigma\rangle$
(Pconc1)	$\langle \mathit{skip} \parallel P, \sigma\rangle \longrightarrow \langle P, \sigma\rangle$
(Pconc2)	$\langle P \parallel \mathit{skip}, \sigma\rangle \longrightarrow \langle P, \sigma\rangle$
(Pconc3)	$\langle P, \sigma\rangle \longrightarrow \langle P', \sigma'\rangle \ \Rightarrow\ \langle P \parallel P'', \sigma\rangle \longrightarrow \langle P' \parallel P'', \sigma'\rangle$
(Pconc4)	$\langle P, \sigma\rangle \longrightarrow \langle P', \sigma'\rangle \ \Rightarrow\ \langle P'' \parallel P, \sigma\rangle \longrightarrow \langle P'' \parallel P', \sigma'\rangle$
(Prec)	$\langle f(\mathit{lfp}\ f), \sigma\rangle \longrightarrow \langle P, \sigma'\rangle \ \Rightarrow\ \langle \mathit{lfp}\ f, \sigma\rangle \longrightarrow \langle P, \sigma'\rangle$

Fig. 7. Small-step operational rules

6 Related Work

Basic setup and definitions of the judgement Compared to this paper, the formalisations in most conventional presentations of rely-guarantee (e.g. [15,18,3,11,2,5]) proceed in a rather different way. They start by giving a grammar that fixes the abstract syntax of programs. Next, they usually give a representation for states (e.g. a state is a

[6] The prefix 'P' in the names of inference rules stands for 'Plotkin' in tribute to [12].

function from identifiers to integers). Programs are then equipped with a small-step operational semantics by choosing a set of inference rules similar to the ones in Figure 7. However, the rules are postulated (i.e. not derived as theorems) and serve to define the small-step judgement. Next, a new small-step judgement is introduced to allow interference by the environment. It uses explicit labels to track whether the program or the environment is responsible for a transition. A popular form of the new judgement is defined by the two inference rules:

$$\langle P,\, \sigma\rangle \xrightarrow{e} \langle P,\, \sigma'\rangle$$
$$\langle P,\, \sigma\rangle \longrightarrow \langle P',\, \sigma'\rangle \;\Rightarrow\; \langle P,\, \sigma\rangle \xrightarrow{p} \langle P',\, \sigma'\rangle$$

A program P is then associated with its execution traces, which are finite or infinite sequences of the form:

$$\langle P_0,\, \sigma_0\rangle \xrightarrow{l_0} \langle P_1,\, \sigma_1\rangle \xrightarrow{l_1} \langle P_2,\, \sigma_2\rangle \ldots$$

where $P_0 = P$, each $l_i \in \Big\{e, p\Big\}$, and each transition in the sequence must be a valid labelled judgement. The rely-guarantee quintuple is then defined in terms of these execution traces of a program. The traces are sometimes summarised by so-called *Aczel traces*, which discard the program components. For example, the Aczel trace of the above execution trace would begin as follows:

$$[(\sigma_0, l_0, \sigma_1), (\sigma_1, l_1, \sigma_2), \ldots$$

Notice that environment steps are incorporated into Aczel traces, and that the labelling helps to determine whether the rely and the guarantee conditions are fulfilled. Alternatively, execution traces can be summarised by so-called *transition traces*. These traces are obtained by dropping all environment transitions from Aczel traces and removing the (now redundant) p-labels. Transition traces look very similar to our traces, but they can be finite or infinite as a result of the operational rules.

This paper proposes a classification of rely-guarantee models into two main groups, but there also exist minor variations in the formal definition of the rely-guarantee judgement within each group. For example:

- Many formalisations place restrictions on rely and guarantee relations such as reflexivity and/or transitivity [8,18,11,2]. Jones originally argued in [8, Chap. 4] that interference should be reflexive and transitive. Subsequent works [18,11] discussed the difficulty of finding transitive conditions in practical examples, and require only reflexivity so that the evaluation of Boolean conditions will automatically satisfy guarantee conditions. Other treatments [15,3] use sets of single-state predicates for rely and guarantee conditions. Dingel [3] showed that such an interference condition corresponds to a binary relation that is both reflexive and transitive.
- Some treatments [2,4,5] require that the pre- and postcondition must be 'stable' with respect to the rely condition in all judgements. An assertion S is stable with respect to interference R iff $R(S) \subseteq S$. The utility of the idea is that it implies $R^*(S) = S$. This means, for example, that a precondition S will still hold when

the program takes its first step. If this step does not change the state (e.g. it evaluates a Boolean condition), then S will still hold after the test. One can also assume the test condition if it is stable under the rely. Likewise, stability can ensure that the environment will preserve the assertion that was established by the last step of the program, thereby turning it into a valid postcondition despite interference.

In contrast to the work discussed before, references [6,1] propose general definitions of the rely-guarantee judgement in algebraic terms. The development in [6] augments a Concurrent Kleene Algebra (CKA) with a set of elements called *invariants* to obtain a rely/guarantee-CKA. It is well known that the set of formal languages with interleaving is a model of CKA (the Isabelle formalisation of this paper also contains a proof). It is hence also the case for trace sets under interleaving. Moreover, by considering each trace set of the form $traces(R)$ for some R to be an invariant, our second model can be viewed as an instance of the abstract model in [6]. It is also an instance of the abstract model in [1], because the sets of traces and invariants also satisfy the laws of *rely-guarantee algebra* proposed there. Equipping rely-guarantee algebra with residuals is briefly considered in [1], and their result (6) can be viewed as an abstract version of our Lemma 6. The trace model that is used to verify examples in [1] is similar to ours, but in order to use laws for Boolean tests such as $test(P); test(Q) = test(P \cap Q)$, it additionally demands that trace sets must be closed under stuttering and mumbling.

Inference rules, their proofs and soundness Figures 1 to 6 include inference rules such as Conjunction and Disjunction that seldom appear in other presentations of rely-guarantee, but which turn out to be useful here for validating other rules (Conjunction helps to strengthen guarantee conditions in the strengthening rules, and Disjunction is used in Section 4 to generalise the rule for sequential composition). Most presentations include weakening rules such as (Jweak) or (Rweak), but they almost never include explicit strengthening rules like (Jstren) or (Rstren). A notable exception is the rule RG-ADJUSTPOST in [16, p. 20], which strengthens the postcondition. This strengthening is often built into the rule for concurrent composition, for example in the rule **Par-I** in [2] and the rule $\|$-I of [9]. In Section 4, the rule (Rconc) strengthens the postcondition of the resulting judgement in a similar way.

As mentioned before, models that are based on operational semantics define the rely-guarantee judgement in terms of execution traces. The proof of the validity of an inference rule such as (Jconc) then directly or indirectly involves the operational rules for concurrent composition such as (Pconc1) through (Pconc4). The resulting proofs can become quite lengthy and involved (see e.g. [2]), but because the definition of the judgement already captures the intended soundness relationship with the operational semantics, there is no need for a separate soundness proof.

Treatments that propose general algebraic definitions for the rely-guarantee judgement prove that inference rules are valid by assuming certain algebraic laws. All models of e.g. a rely/guarantee-CKA must satisfy these laws, and by doing so they automatically gain the rules. The laws thus factor the proofs of the rules into two parts. The proofs that the rules follow from the laws can be surprisingly elegant. Moreover, the laws can have many concrete models (they can also rule out potentially useful models, such as our first one). Each model should explain why the abstract definition of the

judgement is meaningful or appropriate in its context. Although [6,1] do not investigate soundness with repect to operational calculi, each model could also consider it independently. The soundness result in Section 5 demonstrates this for a model (our second one) of both rely/guarantee-CKA and rely-guarantee algebra.

Formalisation in proof assistants Previous formalisations of rely-guarantee in Isabelle/HOL include one by Nieto [11] and one by Armstrong et al. [1]. Nieto's treatment uses a while-language with non-nested concurrent composition and deterministic atomic commands. The language is equipped with an operational semantics, and the rely-guarantee judgement is defined such that the satisfaction of the guarantee condition depends on the precondition and the rely (similar to our first model). Armstrong et al. focus on deriving rely-guarantee rules from algebraic laws. They use an abstract definition of the judgement where the guarantee condition is independent of the precondition and the rely (similar to our second model), and they demonstrate that the algebraic principles can be used to verify while-programs with concurrency.

7 Conclusion

This paper proposes a new classification of semantic models for rely-guarantee into two groups that differ in their treatment of guarantee conditions. To compare them, it constructs an abstract model for each group in a unified setting. The first model supports more powerful inference rules. However, by decoupling the satisfaction of the guarantee from the precondition and the rely, the second model allows algebraic reasoning and an elegant decomposition of the judgement. Both models successfully generalise to the setting where postconditions are binary relations. Both are also sound with respect to operational calculi. Perhaps our classification will have to be extended in the future, but efforts to unify rely-guarantee techniques should at least be flexible enough to accommodate models from both groups described here.

Acknowledgements. This work was supported by the SNSF. Comments by Tony Hoare, Georg Struth and the anonymous referees helped to improve the presentation significantly.

References

1. Armstrong, A., Gomes, V.B., Struth, G.: Algebraic principles for rely-guarantee style concurrency verification tools. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014: Formal Methods, Lecture Notes in Computer Science, vol. 8442, pp. 78–93. Springer International Publishing (2014)
2. Coleman, J.W., Jones, C.B.: A structural proof of the soundness of rely/guarantee rules. J. Log. and Comput. 17(4), 807–841 (Aug 2007), comments refer to the revised version which appeared as technical report CS-TR-1029, University of Newcastle, June 2007.
3. Dingel, J.: A refinement calculus for shared-variable parallel and distributed programming. Formal Asp. Comput. 14(2), 123–197 (2002)

4. Dinsdale-Young, T., Birkedal, L., Gardner, P., Parkinson, M., Yang, H.: Views: compositional reasoning for concurrent programs. In: Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 287–300. POPL '13, ACM, New York, NY, USA (2013)
5. Hayes, I.J., Jones, C.B., Colvin, R.J.: Refining rely-guarantee thinking. Tech. Rep. CS-TR-1334, School of Computing Science, Newcastle University (May 2012)
6. Hoare, C., Möller, B., Struth, G., Wehrman, I.: Concurrent Kleene algebra. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009 - Concurrency Theory, Lecture Notes in Computer Science, vol. 5710, pp. 399–414. Springer Berlin/Heidelberg (2009)
7. Isabelle/HOL proofs: Online at `http://www0.cs.ucl.ac.uk/staff/s.vanstaden/proofs/RG.tgz` (2014)
8. Jones, C.B.: Development Methods for Computer Programs including a Notion of Interference. Ph.D. thesis, Oxford University (June 1981), printed as: Programming Research Group, Technical Monograph 25
9. Jones, C.B.: Balancing expressiveness in formal approaches to concurrency. Formal Aspects of Computing (submitted) (2013)
10. Morgan, C.: The specification statement. ACM Trans. Program. Lang. Syst. 10, 403–419 (July 1988)
11. Nieto, L.P.: The rely-guarantee method in Isabelle/HOL. In: Proceedings of the 12th European Symposium on Programming. pp. 348–362. ESOP'03, Springer-Verlag, Berlin, Heidelberg (2003)
12. Plotkin, G.D.: A structural approach to operational semantics. Tech. Rep. DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark (September 1981)
13. de Roever, W.P., de Boer, F.S., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods, Cambridge Tracts in Theoretical Computer Science, vol. 54. Cambridge University Press (2001)
14. van Staden, S.: Constructing the views framework. In: Naumann, D. (ed.) Unifying Theories of Programming, Lecture Notes in Computer Science, vol. 8963, pp. 62–83. Springer International Publishing (2015)
15. Stirling, C.: A generalization of Owicki-Gries's Hoare logic for a concurrent while language. Theor. Comput. Sci. 58, 347–359 (1988)
16. Vafeiadis, V.: Modular fine-grained concurrency verification. Tech. Rep. UCAM-CL-TR-726, University of Cambridge, Computer Laboratory (July 2008)
17. Wickerson, J., Dodds, M., Parkinson, M.: Explicit stabilisation for modular rely-guarantee reasoning. In: Gordon, A.D. (ed.) Programming Languages and Systems, Lecture Notes in Computer Science, vol. 6012, pp. 610–629. Springer Berlin Heidelberg (2010)
18. Xu, Q., de Roever, W.P., He, J.: The rely-guarantee method for verifying shared variable concurrent programs. Formal Asp. Comput. 9(2), 149–174 (1997)