On Rely-Guarantee Reasoning

Stephan van Staden

June 29, 2015
Overview

Introduction
  Basics of Rely-Guarantee
  Overview of the paper
    Two techniques for constructing Rely-Guarantee models
    Soundness results

The “traditional” technique
  Example of a model
  Some valid rules
  Relationship with Hoare logic

The “recent” technique
  Example of a model
  Some valid rules
  Special properties and comparison with “traditional” technique

Soundness of both techniques

Conclusion
Rely-Guarantee 101

Extension of Hoare logic (HL) for reasoning about concurrency.

Makes interference explicit in specifications – rely & guarantee. They are binary relations on states that summarise the state transformations of execution steps.

Judgements have the form: \textit{Pre Rely} \{\textit{Prog}\} \textit{Guar Post}.

\textit{Rely} relation says what interference \textit{Prog} can tolerate while still satisfying the \textit{Pre-Post} spec.

\textit{Guar} relation says what interference \textit{Prog} can inflict on its concurrent environment.

Example: \( S \times S \{\text{skip}\} \text{ Id } S. \)
Rely-Guarantee 101 (continued)

RG offers a compositional rule for concurrency:

\[
S_1 R_1 \{P\} G_1 S'_1 \land S_2 R_2 \{Q\} G_2 S'_2 \land G_1 \subseteq R_2 \land G_2 \subseteq R_1 \Rightarrow (S_1 \cap S_2) (R_1 \cap R_2) \{P \parallel Q\} (G_1 \cup G_2) (S'_1 \cap S'_2).
\]
RG offers a compositional rule for concurrency:
\[ S_1 R_1 \{ P \} G_1 S'_1 \land S_2 R_2 \{ Q \} G_2 S'_2 \land G_1 \subseteq R_2 \land G_2 \subseteq R_1 \Rightarrow (S_1 \cap S_2) (R_1 \cap R_2) \{ P \parallel Q \} (G_1 \cup G_2) (S'_1 \cap S'_2). \]

Every valid RG quintuple yields a valid HL triple:
\[ S R \{ P \} G S' \Rightarrow S \{ P \} S'. \]

Enriched specs conquer concurrency!

In fact, HL is embedded in RG:
\[ S \{ P \} S' \Leftrightarrow S \bot \{ P \} \top S'. \]
Generalising HL to RG involves a *creative leap*.

At least *two distinct techniques* for this exist that differ in their treatment of guarantee conditions:

- “Traditional” technique (Jones, Stirling, Dingel, Coleman, …) uses a weaker RG judgement and supports more general rules for atomic commands and disjunction.
- “Recent” technique (Hayes, CKA work ...) has a stronger RG judgement that can be decomposed into smaller constructs. It facilitates nice algebraic/refinement-style proofs.

Despite their differences, both techniques satisfy the properties of the previous slides.

The paper also gives a new proof of the soundness of both techniques w.r.t. operational calculi.
“Traditional” technique: example model

Informal meaning of $S R \{P\} G S'$:

if

- program $P$ is executed in a state which satisfies $S$, and
- every environment step satisfies $R$,

then

- every step of $P$ satisfies $G$, and
- if the execution terminates, then the final state satisfies $S'$. 

Formally, require each trace of $P$ to behave properly (a trace is a list of state pairs that describe $P$'s ability to transform states):

$$S R \{P\} G S' \overset{\text{def}}{=} \forall t \in P : \text{rg-trace } S R t G S'$$

$$\text{rg-trace } S R \{\sigma, \sigma'\} : t G S' \overset{\text{def}}{=} \sigma \in R^* (S) \Rightarrow (\sigma, \sigma') \in G \land \text{rg-trace } \{\sigma'\} R t G S'$$
“Traditional” technique: example model

Informal meaning of $SR\\{P\}\ G S'$:
if
  ▶ program $P$ is executed in a state which satisfies $S$, and
  ▶ every environment step satisfies $R$,
then
  ▶ every step of $P$ satisfies $G$, and
  ▶ if the execution terminates, then the final state satisfies $S'$.

Formally, require each trace of $P$ to behave properly (a trace is a list of state pairs that describe $P$'s ability to transform states):

$$SR\{P\}\ G S' \overset{\text{def}}{=} \forall t \in P : \text{rg-trace } SR t G S'$$

$$\text{rg-trace } SR [] G S' \overset{\text{def}}{=} R^*(S) \subseteq S'$$

$$\text{rg-trace } SR ((\sigma, \sigma') : t) G S' \overset{\text{def}}{=} \sigma \in R^*(S) \Rightarrow (\sigma, \sigma') \in G \land \text{rg-trace } \{\sigma\} R t G S'.$$
“Traditional” technique: examples of valid rules

Concurrency:
\[ S_1 R_1 \{P\} G_1 S_1' \land S_2 R_2 \{Q\} G_2 S_2' \land G_1 \subseteq R_2 \land G_2 \subseteq R_1 \Rightarrow (S_1 \cap S_2) (R_1 \cap R_2) \{P \parallel Q\} (G_1 \cup G_2) (S_1' \cap S_2'). \]

Weakening:
\[ S_1 R_1 \{P\} G_1 S_1' \land S_2 \subseteq S_1 \land R_2 \subseteq R_1 \land G_1 \subseteq G_2 \land S_1' \subseteq S_2' \Rightarrow S_2 R_2 \{P\} G_2 S_2'. \]

Atomic commands:
\[ rel(a) \cap (R^*(S)) \times \Sigma \subseteq G \land (R^*; rel(a); R^*)(S) \subseteq S' \Rightarrow S R \{a\} G S'. \]

Disjunction:
\[ (\forall S \in Y : S R \{P\} G S') \Rightarrow (\bigcup Y) R \{P\} G S'. \]
“Traditional” technique: relationship with HL

\[
S \ R \ \{P\} \ G \ S' \quad \Rightarrow \quad S \perp \ {P} \uplus S' \ \text{holds by Weakening.}
\]

\[
S \perp \ \{P\} \uplus S' \quad \iff \quad S \ {P} \ S' \ \text{is true, and in fact we can prove a more general theorem for arbitrary rely conditions:}
\]

\[
S \ R \ \{P\} \uplus S' \quad \iff \quad S \ {P \parallel traces(R)} \ S'.
\]
“Traditional” technique: relationship with HL

\[ S R \{P\} G S' \implies S \perp \{P\} \triangleright S' \] holds by Weakening.

\[ S \perp \{P\} \triangleright S' \iff S \{P\} S' \] is true, and in fact we can prove a more general theorem for arbitrary rely conditions:

\[ S R \{P\} \triangleright S' \iff S \{P \parallel \text{traces}(R)\} S' \].

No straightforward way to generalise this to arbitrary guarantee conditions.
Why? Because the satisfaction of the guarantee depends on the program, the precondition, and the rely condition.
E.g. Coleman & Jones use the judgement \( \{S, R\} \models P \text{ within } G \).
“Recent” technique: example model

\[ S R \{P\} G S' \overset{\text{def}}{=} S\{P \parallel \text{traces}(R)\} S' \land P \subseteq \text{traces}(G). \]

The satisfaction of the guarantee depends on the program only, irrespective of the precondition and the rely.
“Recent” technique: example model

\[ SR \{P\} G S' \overset{\text{def}}{=} S\{P \parallel \text{traces}(R)\} S' \land P \subseteq \text{traces}(G). \]

The satisfaction of the guarantee depends on the program only, irrespective of the precondition and the rely.

Informal meaning of \( SR \{P\} G S' \):

- every step of program \( P \) satisfies \( G \), and
- if
  - \( P \) is executed in a state which satisfies \( S \), and
  - every environment step satisfies \( R \),
  then
    - if the execution terminates, then the final state satisfies \( S' \).

Stronger judgement than in the “traditional” technique.
“Recent” technique: examples of valid rules

Concurrency and Weakening as before.

Atomic commands:
\[ \text{rel}(a) \subseteq G \land (R^*; \text{rel}(a); R^*)(S) \subseteq S' \Rightarrow S R \{a\} G S'. \]

Disjunction:
\[ Y \neq \emptyset \land (\forall S \in Y : S R \{P\} G S') \Rightarrow (\bigcup Y) R \{P\} G S'. \]

In contrast to the “traditional” technique, \( \emptyset R \{P\} G S' \) is not valid for arbitrary \( R, P, G, S' \! \). However, \( \emptyset R \{P\} \vdash S' \) is valid and the expected relationships with HL hold.

Proofs of rules (and judgements) can be formulated i.t.o. HL and refinement.
“Recent” technique: decomposing the quintuple

Let $P \parallel P'$ be the largest (i.e. most nondeterministic) program whose concurrent composition with $P$ is included in $P'$, i.e. $P'' \parallel P \subseteq P' \iff P'' \subseteq P - \parallel P'$.

$$\text{rely } R \; P \; \overset{\text{def}}{=} \; \text{traces}(R) - \parallel P.$$ $$\text{guar } G \; P \; \overset{\text{def}}{=} \; \text{traces}(G) \cap P.$$  

The rely and guar operators have nice algebraic properties.

They can decompose the RG quintuple of the “recent” technique: $S \; R \; \{P\} \; G \; S' \iff P \subseteq \text{guar } G \; (\text{rely } R \; [S, S'])$.

So RG rules and judgements follow from the properties of simple operators. Delightful refinement-style derivations (see work by Ian Hayes et al.).
Soundness

Define a big-step operational judgement:

\[ \langle P, \sigma \rangle \rightarrow \sigma' \ \overset{\text{def}}{=} \ \exists t \in \text{IF-traces-ending-in}(\sigma) : \exists t' \in \text{IF-traces-ending-in}(\sigma') : \{ t \} ; P \supseteq \{ t' \}. \]

All the familiar big-step operational rules are theorems, e.g.

\[ \langle P, \sigma \rangle \rightarrow \sigma' \land \langle P', \sigma' \rangle \rightarrow \sigma'' \Rightarrow \langle P ; P', \sigma \rangle \rightarrow \sigma''. \]
Soundness

Define a big-step operational judgement:
\[ \langle P, \sigma \rangle \rightarrow \sigma' \overset{\text{def}}{=} \exists t \in IF\text{-}traces\text{-}ending\text{-}in(\sigma) : \exists t' \in IF\text{-}traces\text{-}ending\text{-}in(\sigma') : \{ t \}; P \supseteq \{ t' \}. \]

All the familiar big-step operational rules are theorems, e.g.
\[ \langle P, \sigma \rangle \rightarrow \sigma' \land \langle P', \sigma' \rangle \rightarrow \sigma'' \Rightarrow \langle P; P', \sigma \rangle \rightarrow \sigma''. \]

Relationship with Hoare triple:
\[
S \{ P \} S' \iff (\forall \sigma \in S : \forall \sigma' : \langle P, \sigma \rangle \rightarrow \sigma' \Rightarrow \sigma' \in S').
\]

So the RG judgements of both techniques are sound w.r.t. big-step execution:
\[
S R \{ P \} G S' \Rightarrow (\forall \sigma \in S : \forall \sigma' : \langle P, \sigma \rangle \rightarrow \sigma' \Rightarrow \sigma' \in S').
\]
Soundness (continued)

The big-step judgement has no compositional rule for concurrency. Define a small-step operational judgement:
\[ \langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \overset{\text{def}}{=} \exists Q \in \text{Actions} : P \supseteq Q ; P' \land \langle Q, \sigma \rangle \rightarrow \sigma'. \]

All the familiar small-step rules are theorems, including:
\[ \langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \Rightarrow \langle P \parallel P'', \sigma \rangle \rightarrow \langle P' \parallel P'', \sigma' \rangle. \]
\[ \langle P, \sigma \rangle \rightarrow \langle \text{skip}, \sigma' \rangle \Rightarrow \langle P \parallel P'', \sigma \rangle \rightarrow \langle P'', \sigma' \rangle. \]
Soundness (continued)

The big-step judgement has no compositional rule for concurrency. Define a small-step operational judgement:
\[
\langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \overset{\text{def}}{=} \exists Q \in \text{Actions} : P \supseteq Q \land P' \land \langle Q, \sigma \rangle \rightarrow \sigma'.
\]

All the familiar small-step rules are theorems, including:
\[
\langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \Rightarrow \langle P \parallel P'', \sigma \rangle \rightarrow \langle P' \parallel P'', \sigma' \rangle.
\]
\[
\langle P, \sigma \rangle \rightarrow \langle \text{skip}, \sigma' \rangle \Rightarrow \langle P \parallel P'', \sigma \rangle \rightarrow \langle P'', \sigma' \rangle.
\]

Relationship with the big-step judgement:
\[
\langle P, \sigma \rangle \rightarrow^{*} \langle \text{skip}, \sigma' \rangle \Rightarrow \langle P, \sigma \rangle \rightarrow \sigma'. \text{ (Compare RG & HL.)}
\]
Soundness (continued)

The big-step judgement has no compositional rule for concurrency.
Define a small-step operational judgement:
\[ \langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \overset{\text{def}}{=} \exists Q \in \text{Actions} : P \supseteq Q ; P' \land \langle Q, \sigma \rangle \rightarrow \sigma'. \]

All the familiar small-step rules are theorems, including:
\[ \langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \Rightarrow \langle P \parallel P'', \sigma \rangle \rightarrow \langle P' \parallel P'', \sigma' \rangle. \]
\[ \langle P, \sigma \rangle \rightarrow \langle \text{skip}, \sigma' \rangle \Rightarrow \langle P \parallel P'', \sigma \rangle \rightarrow \langle P'', \sigma' \rangle. \]

Relationship with the big-step judgement:
\[ \langle P, \sigma \rangle \rightarrow^* \langle \text{skip}, \sigma' \rangle \Rightarrow \langle P, \sigma \rangle \rightarrow \sigma'. \] (Compare RG & HL.)

So both RG judgements are sound w.r.t. small-step execution:
\[ SR \{ P \} \ G S' \Rightarrow (\forall \sigma \in S : \forall \sigma' : \langle P, \sigma \rangle \rightarrow^* \langle \text{skip}, \sigma' \rangle \Rightarrow \sigma' \in S'). \]

The argument does not depend on the (abstract) syntax of programs, the chosen set of Actions, or a specific selection of inference rules.
Recap

Picked a semantic model (of traces) rich enough to model

▶ relevant operators, such as ;, ||, choice, recursion
▶ various judgements, such as RG (two flavours), HL, big-step and small-step execution.

It then established

▶ interesting relationships between the judgements, e.g. weaker/stronger ones, soundness theorems, . . .
▶ valid inference rules for each judgement.
Recap

Picked a semantic model (of traces) rich enough to model
▶ relevant operators, such as ;, ||, choice, recursion
▶ various judgements, such as RG (two flavours), HL, big-step and small-step execution.

It then established
▶ interesting relationships between the judgements, e.g. weaker/stronger ones, soundness theorems, ...
▶ valid inference rules for each judgement.

This shed some light on semantic issues:
▶ It facilitated a comparison between two flavours of RG.
▶ The soundness results transfer to all concrete settings where the judgements are defined by inference rules which are theorems in this development. (Reason: the soundness theorem will remain valid for stronger judgements.)
Conclusions

There are (at least) two distinct ways to equip HL with interference specs.

Different treatments of guarantee conditions lead to differences in:

- The rules for atomic commands and disjunction.
- The style of reasoning: trace-based vs. algebraic metatheory; quintuple judgements vs. \textit{rely} & \textit{guar} constructs and refinement.

Yet despite these differences:

- They share mostly the same rules (e.g. Concurrency, Weakening, . . . ).
- They enjoy similar relationships with HL
- and are hence both sound w.r.t. familiar operational calculi for the same simple reason.