Hoare-style Reasoning from Multiple Contracts*

Olaf Owe, Toktam Ramezanifarkhani, and Elahe Fazeldehkordi

Department of Informatics, University of Oslo, Norway
{olaf, toktamr, elahefa}@ifi.uio.no

Abstract. Modern software is often developed with advanced mechanisms for code reuse. A software module may build on other software modules or libraries where the source code is not available. And even if the source code is known, the binding mechanism may be such that the binding of methods is not known at verification time, and thus the underlying reused code cannot be determined. For example, in delta-oriented programming the binding of methods depends on the ordering of deltas in each product, making modular reasoning non-trivial. Similar problems occur with traits and subclassing. Reasoning inside a module must then be based on partial knowledge of the methods, typically given by contracts in the form of pairs of pre- and post-conditions, and one may not derive new properties by re-verification of the (unavailable) source code.
In the setting of Hoare logic, this gives some challenges to general rules for adaptation that goes beyond traditional systems for Hoare logic. We develop a novel way of reasoning from multiple contracts, which makes the traditional adaptation rules superfluous. The problem we address does not depend on the choice of programming language. We therefore focus on general rules rather than statement-specific rules. We show soundness and completeness for the suggested rules.

Keywords: Hoare Logic; Multiple Contracts; Contract-Based Verification; Contract-Based Specification; Adaptation; Unavailable Source Code; Soundness; Completeness.

1 Introduction

We consider an open world program development environment where a program may depend on program parts that are not yet known or available. For instance, in software product lines based on delta-oriented programming[25] the binding of methods depends on the ordering of deltas, which may differ from product to product. In a method definition in a delta, the special notation *original* binds to the previous version of the same method in the delta closest in the delta-ordering of the product. Thus for a given delta the binding of method calls is not known, and reasoning about the delta must be based on the contracts about methods of underlying modules. Similar complications are caused by *traits* [27,12]. And for class-based programs with inheritance and late binding, the binding of calls (including self calls) may not be known at verification time. In these settings, it is important to be able to reason about the program in a modular fashion, based on contracts representing partial knowledge of methods for which the binding is not known.

* This work is supported by the Norwegian NRC projects *IoTSec: Security of IoT* and *DiversIoT*.

Contract-based specification is well known from the work on design by contract by Meyer [18] and has been adopted in several languages and frameworks, including Eiffel [19] and SPEC# [3]. In this paper we will focus on contract-based verification, i.e., reasoning from contracts rather than reasoning about verifying contracts. We assume that a given program part (such as a method) is specified by a set of contracts, and consider a Hoare style reasoning framework. A contract may give partial knowledge of a program part, and we assume that a given program part may have *several contracts*, possibly accumulated over time as more knowledge becomes available.

A Hoare triple $\{P\}\ \mathrm{s}\ \{Q\}$ characterizes the effect of a program (part) s by means of a precondition P and a postcondition Q, where both P and Q may refer to program variables as well as additional non-program variables, so-called *logical variables* [15]. The intuitive meaning of this triple is that if the program s is executed from a state ("the prestate") where P holds and the execution terminates, then Q will hold in the final state ("the poststate"). The pair $(\mathrm{P},\ \mathrm{Q})$ is called a *contract* for s.

Example 1. For instance, the contract $(\mathrm{x} < 0,\ \mathrm{x} < 0)$ expresses that if the program variable x is negative in the prestate, it will also be negative in the poststate (if any), and the contract $(\mathrm{x} > 0,\ \mathrm{x} > 0)$ similarly restricts the final value of x when positive in the prestate. The contract $(\mathrm{x} = 0,\ \mathrm{x} = 0)$ characterizes the final value of x when its starting value is 0. All three contracts are examples of invariants. Using a logical variable y', the combination of the contracts $(\mathrm{x} < 0 \wedge \mathrm{y} = \mathrm{y}',\ \mathrm{y} = \mathrm{y}')$ and $(\mathrm{x} \geq 0 \wedge \mathrm{y} = \mathrm{y}',\ \mathrm{y} = \mathrm{y}' + \mathrm{x})$ gives full information about the final value of the program variable y.

The combination of all five example contracts above should imply the two following contracts (of which the latter is an invariant):

$$(\mathrm{y} = \mathrm{y}',\ \mathrm{x} < 0 \wedge \mathrm{y} = \mathrm{y}' \vee \mathrm{x} \geq 0 \wedge \mathrm{y} = \mathrm{y}' + \mathrm{x}) \tag{1}$$

$$(\mathrm{x} = 0 \wedge \mathrm{y} = \mathrm{y}',\ \mathrm{x} = 0 \wedge \mathrm{y} = \mathrm{y}') \tag{2}$$

Traditionally, Hoare logics are used to deal with one contract per program. An occurrence of s may have a program environment which needs some adjustment or adaptation of the contract. The simplest way of adjustment is provided by the well-known Rule of Consequence:

$$\frac{\mathrm{P}' \Rightarrow \mathrm{P} \qquad \{P\}\ \mathrm{s}\ \{Q\} \qquad \mathrm{Q} \Rightarrow \mathrm{Q}'}{\{P'\}\ \mathrm{s}\ \{Q'\}}\ (\textsc{consequence})$$

However, this rule is not suitable when s is used in a context where the assertions talk about more variables than the contract of s, nor when the precondition of the contract is not satisfied. For instance, from the contracts given above, we may not derive contract 1 nor 2 by the consequence rule alone. Moreover, if we use the conjunction rule on the contracts $(\mathrm{x} > 0,\ \mathrm{x} > 0)$, $(\mathrm{x} < 0,\ \mathrm{x} < 0)$, and $(\mathrm{x} = 0,\ \mathrm{x} = 0)$, we get false in the precondition. If we use the disjunction rule, we get true in the postcondition. Neither gives any information, so we need better ways to combine the information in several contracts.

Adaptation rules give more freedom in the reasoning, by allowing different preconditions and postconditions for different reuses (i.e., occurrences) of s. For instance, we may try to prove the example above, or we may try to find a precondition P such that

$\{P\}$ s $\{x = x' \wedge y = y' + x\}$ is satisfied for s as in the example. The original adaptation rule suggested by Hoare [16] can be formulated as

$$\frac{\{P\}\ \mathrm{s}\ \{Q\}}{\{\exists \overline{z}.\ (P \wedge \forall \overline{w}.\ (Q \rightarrow R))\}\ \mathrm{s}\ \{R\}}\ (\textsc{adaptation})$$

where $\overline{\mathrm{w}}$ is the list of program variables that may be changed by s and $\overline{\mathrm{z}}$ is list of logical variables in (P, Q) but not in R. As shown in [20], this adaptation rule is sound and relatively complete, although the precondition is not the weakest possible formulation. As discussed in [1], the precondition may be weakened to $\forall \overline{\mathrm{w}}'(\forall \overline{\mathrm{z}}.\ \mathrm{P} \Rightarrow \mathrm{Q}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'}) \Rightarrow \mathrm{R}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'})$, where $\mathrm{Q}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'}$ denotes Q with all free occurrences of $\overline{\mathrm{w}}$ replaced by $\overline{\mathrm{w}}'$ (given that the lists $\overline{\mathrm{w}}$ and $\overline{\mathrm{w}}'$ have the same length). In this precondition, $\overline{\mathrm{z}}$ are the logical variables in P and/or Q.

However, we are still not able to handle multiple contracts, and for the example above, we may still not derive contract 1 nor 2. Special adaptation rules considering the case of multiple contracts have been suggested, and will be discussed in the next section.

The contribution of this paper is to reconsider Hoare-style reasoning supporting reasoning from contracts when the program text is not available or known, allowing multiple contracts. Traditionally, completeness of Hoare logics assume that all parts of the program text are available, and thus may be used to re-verify different contracts when needed. We suggest a new approach for adaptation. We first introduce a rule called the *normalization rule*, which is simpler then the adaptation rule. We then introduce a generalized version of the rule, called the *generalized normalization rule*, for dealing with multiple contracts. The combination of the generalized normalization rule and the consequence rule forms a complete set of general reasoning rules. We prove soundness and completeness of these rules, and show that the (most expressive) adaptation rule and its generalization to multiple contracts, as well as other general rules, can be derived easily.

2 Reasoning about Multiple Contracts

In order to use a contract in program reasoning, one needs to know an overapproximation of the set of program variables that may be changed by the program, called the *var* set. This may be part of a contract. However, when considering delta-oriented programming, or object-oriented inheritance, a redefined method may involve variables that were not known when the original contract was formulated. This makes it impossible to restrict the *var* set for contracts that are intended to be reused/inherited. In this case an underestimation of the set of read-only variables can be valuable [8]. Instead of letting the contract specify the *var* set, one may overapproximate the set of program variables that may change for each call. As different choices of programming languages may require different kinds of specification of *var* sets, we will not impose a fixed way of specifying *var* sets, but assume that a *var* set $\overline{\mathrm{w}}$ and a set of read-only variables $\overline{\mathrm{v}}$ can be determined statically for each occurrence of a program/call specified by a contract.

Thus, for each occurrence of a program we let $\overline{\mathrm{w}}$ denote the (finite) set of program variables that may be changed by a program s, and $\overline{\mathrm{v}}$ denote the complementary set of program variables that may not be changed by s. In addition to the program variables, a contract may talk about logical variables, which are here denoted by the symbol z or by primed versions of the program variables.

Consider the case that several contracts may be assumed (or have been proved) for a given program, or method body, s. This can be expressed by a set of triples $\{P_i\}\ \mathrm{s}\ \{Q_i\}$ for $1 \leq \mathrm{i} \leq \mathrm{N}$. How can we make use of this knowledge? The following adaptation rule expresses how to exploit the knowledge of several such triples. For any assertion R the rule allows us to derive a precondition of s with exactly R as the postcondition:

$$\frac{\{P_i\}\ \mathrm{s}\ \{Q_i\} \quad \text{for } 1 \leq \mathrm{i} \leq \mathrm{N}}{\{\forall \overline{w}'\,.\,(\bigwedge_i \forall \overline{z_i}\,.\,(P_i \Rightarrow Q_i{}^{\overline{w}}_{\overline{w}'})) \Rightarrow R^{\overline{w}}_{\overline{w}'}\}\ \mathrm{s}\ \{R\}} \quad \text{(GENERALIZED ADAPTATION)}$$

where $\overline{\mathrm{w}}'$ are fresh logical variables, $\overline{\mathrm{z}_\mathrm{i}}$ the logical variables in $(\mathrm{p_i}, \mathrm{q_i})$, and $\overline{\mathrm{w}}$ variables that may be updated by the method. A similar rule for right-constructive reasoning was suggested in [29].

Example 2. Given the Hoare triples

$$\{x \geq 0\}\ \mathrm{s}\ \{x \geq 0\} \quad \text{and} \quad \{x \leq 0\}\ \mathrm{s}\ \{x \leq 0\}$$

How can we combine this knowledge in one Hoare triple? The generalized adaptation rule gives $\{\forall x'\,.\,(x \geq 0 \Rightarrow x' \geq 0) \wedge (x \leq 0 \Rightarrow x' \leq 0) \Rightarrow R^x_{x'}\}\ \mathrm{s}\ \{R\}$. Taking R as $(\mathrm{x}' \geq 0 \Rightarrow \mathrm{x} \geq 0) \wedge (\mathrm{x}' \leq 0 \Rightarrow \mathrm{x} \leq 0)$, the precondition becomes

$$\forall \mathrm{x}'.((\mathrm{x} \geq 0 \Rightarrow \mathrm{x}' \geq 0) \wedge (\mathrm{x} \leq 0 \Rightarrow \mathrm{x}' \leq 0)) \Rightarrow ((\mathrm{x}' \geq 0 \Rightarrow \mathrm{x}' \geq 0) \wedge (\mathrm{x}' \leq 0 \Rightarrow \mathrm{x}' \leq 0))$$

which is implied by $\mathrm{x} = \mathrm{x}'$. By the consequence rule we may then derive

$$\{x = x'\}\ \mathrm{s}\ \{(x' \geq 0 \Rightarrow x \geq 0) \wedge (x' \leq 0 \Rightarrow x \leq 0)\}$$

Thus, the generalized adaptation rule is able to handle this example, as well as contracts 1 and 2. However, it has some drawbacks. The rule is quite complicated. In particular, the precondition of the conclusion has non-trivial nesting of implications and quantifiers. The quantifier on each $\overline{\mathrm{z}}_\mathrm{i}$ is essentially an existential quantification (when lifted out of the implicant). In addition the rule is specialized to the setting of left-constructive reasoning (for a given postcondition R), and the usage of logical variables in R leads to rather complicated reasoning. A typical case appears when showing that the precondition of a given contract (for a given program) implies the precondition generated by left-constructive rules; in this situation the generalized adaptation rule leads to reasoning with existential quantifiers that can be non-trivial (for humans as well as automated tools).

2.1 Semantics

The validity of a Hoare triple $\{P\}\mathrm{s}\{Q\}$ is denoted $\models\{P\}\mathrm{s}\{Q\}$ and is defined as follows:

$$\models\{P\}\ \mathrm{s}\ \{Q\} \equiv \forall \overline{\mathrm{z}}, \overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''\,.\,(\overline{\mathrm{v}}, \overline{\mathrm{w}}' [\![\mathrm{s}]\!] \overline{\mathrm{w}}'') \wedge \mathrm{P}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow \mathrm{Q}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''}$$

where $[\![\mathrm{s}]\!]$ is the input/output relation defined by the program s, and where $\overline{\mathrm{z}}$ is the list of logical variables occurring in P and/or Q, $\overline{\mathrm{v}}$ is the list of program variables that may not change in s, $\overline{\mathrm{w}}$ is the list of program variables that may change in s, and $\overline{\mathrm{w}}'$ and $\overline{\mathrm{w}}''$ are lists of logical variables distinct from $\overline{\mathrm{z}}$ (of same length and types as $\overline{\mathrm{w}}$).

The logical variables $\overline{\mathrm{w}}'$ denote the prestate values of variables that may change in s and $\overline{\mathrm{w}}''$ that of the poststate. The semantical meaning of a program s is here formalized as an input/output relation, which for a given state (i.e., values of $\overline{\mathrm{v}}$ and $\overline{\mathrm{w}}$) gives the possible output states, restricted to variables that may change ($\overline{\mathrm{w}}$). Thus, $\overline{\mathrm{v}}, \overline{\mathrm{w}}' [\![\mathrm{s}]\!] \overline{\mathrm{w}}''$ expresses that if s starts in the prestate $\overline{\mathrm{v}}, \overline{\mathrm{w}}'$, then $\overline{\mathrm{v}}, \overline{\mathrm{w}}''$ is the poststate for some terminating execution of s. If no output states exists the program does not terminate normally. (Non-deterministic non-termination is then not captured, but since we deal with partial correctness, this can be ignored.) We assume a fixed interpretation of data types and related functions.

2.2 Normalization Rules

For $\overline{\mathrm{z}}, \overline{\mathrm{v}}, \overline{\mathrm{w}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''$ as above, the definition of validity gives that $\models\{P\}\ \mathrm{s}\ \{Q\}$ is the same as

$$\forall \overline{\mathrm{z}}, \overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}'' \,.\, (\overline{\mathrm{v}}, \overline{\mathrm{w}}' [\![\mathrm{s}]\!] \overline{\mathrm{w}}'') \wedge (\overline{\mathrm{w}} = \overline{\mathrm{w}}')^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow (\mathrm{P}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow \mathrm{Q})^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''}$$

since $\mathrm{P}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'}$ has no free w and since $(\overline{\mathrm{w}} = \overline{\mathrm{w}}')^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'}$ is equivalent to true. This can be reformulated as

$$(\overline{\mathrm{v}}, \overline{\mathrm{w}}' [\![\mathrm{s}]\!] \overline{\mathrm{w}}'') \wedge (\overline{\mathrm{u}} = \overline{\mathrm{u}}')^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow (\forall \overline{\mathrm{z}} \,.\, \mathrm{P}^{\overline{\mathrm{u}}}_{\overline{\mathrm{u}}'} \Rightarrow \mathrm{Q})^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''} \tag{3}$$

for all $\overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''$, where $\overline{\mathrm{u}}$ is a sublist of $\overline{\mathrm{w}}$ that includes all program variables occurring in P, and $\overline{\mathrm{u}}'$ is the corresponding sublist of $\overline{\mathrm{w}}'$. Then $\mathrm{P}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'}$ is the same as $\mathrm{P}^{\overline{\mathrm{u}}}_{\overline{\mathrm{u}}'}$. By the definition of validity, equation 3 is the same as $\models\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\forall \overline{z} \,.\, P^{\overline{u}}_{\overline{u}'} \Rightarrow Q\}$. Thus we have proved

$$\models\{P\}\ \mathrm{s}\ \{Q\} \Leftrightarrow \models\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\forall \overline{z} \,.\, P^{\overline{u}}_{\overline{u}'} \Rightarrow Q\}$$

for $\overline{\mathrm{u}}'$ not occurring in P nor Q. And thereby, we have proved the validity of the rule

$$\frac{\{P\}\ \mathrm{s}\ \{Q\}}{\{\overline{\mathrm{u}} = \overline{\mathrm{u}}'\}\ \mathrm{s}\ \{\forall \overline{\mathrm{z}} \,.\, \mathrm{P}^{\overline{\mathrm{u}}}_{\overline{\mathrm{u}}'} \Rightarrow \mathrm{Q}\}} \text{(NORMALIZATION)}$$

where $\overline{\mathrm{z}}$ is the list of logical variables in (P, Q), $\overline{\mathrm{u}}$ is a list of program variables including those in P, and $\overline{\mathrm{u}}'$ is fresh (i.e., does not occur in P nor Q), and where the double line means that the rule can be used both ways, which is the case here since the validity of the premise is equivalent to that of the conclusion. We may choose $\overline{\mathrm{u}}$ as exactly the program variables in P (in order to get simple preconditions), or as $\overline{\mathrm{w}}$ (in order to easily compare different normalized triples). When no variable in $\overline{\mathrm{w}}$ occur in P, the precondition $\overline{\mathrm{u}} = \overline{\mathrm{u}}'$ may be replaced by *true*, and $\mathrm{P}^{\overline{\mathrm{u}}}_{\overline{\mathrm{u}}'}$ reduces to P.

This rule is useful since the precondition is incorporated in the postcondition, and thus *we can compare two Hoare triples by simply comparing their normalized postconditions* (choosing $\overline{\mathrm{u}}$ as $\overline{\mathrm{w}}$ so that the preconditions are the same). The rule basically

expresses the strongest postcondition of a program s. Since the rule may be used backwards, we implicitly have the rule

$$\frac{\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\forall \overline{z}.\ P^{\overline{u}}_{\overline{u}'} \Rightarrow Q\}}{\{P\}\ \mathrm{s}\ \{Q\}} \quad \text{(BACKWARD NORMALIZATION)}$$

where $\overline{\mathrm{u}}$ is a list of program variables including those in P, $\overline{\mathrm{u}}'$ is fresh (i.e., not occurring in P or Q) and $\overline{\mathrm{z}}$ is a list of logical variables. In this rule, there is no restriction on $\overline{\mathrm{z}}$, apart from being a list of logical variables disjoint from $\overline{\mathrm{u}}'$.

The generalized normalization rule.
When dealing with a number of contracts for the same program, we propose the following generalization of the normalization rule, which combines the information in a set of contracts for a given program s into one normalized triple:

$$\frac{\{P_i\}\ \mathrm{s}\ \{Q_i\} \text{ for each } \mathrm{i} \in \mathrm{I}}{\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\bigwedge_{i \in I} \forall \overline{z_i}.\ P_i{}^{\overline{u}}_{\overline{u}'} \Rightarrow Q_i\}} \quad \text{(GENERALIZED NORMALIZATION)}$$

where $\overline{\mathrm{u}}$ includes the program variables in any $\mathrm{P_i}$.

Since the rule is two-way, there is no information loss in applying the rule. When the rule is used forwards, knowledge from multiple contracts is combined, and when used backwards, the individual contracts can be recreated.

Proof of Soundness. By the same argumentation as above, the validity of each premise i can be expressed as

$$(\overline{\mathrm{v}}, \overline{\mathrm{w}}' [\![\mathrm{s}]\!] \overline{\mathrm{w}}')' \wedge (\overline{\mathrm{u}} = \overline{\mathrm{u}}')^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow (\forall \overline{\mathrm{z_i}}.\ \mathrm{P_i}^{\overline{\mathrm{u}}}_{\overline{\mathrm{u}}'} \Rightarrow \mathrm{Q_i})^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''}$$

with $\overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''$ universally quantified, and thus we have

$$(\overline{\mathrm{v}}, \overline{\mathrm{w}}' [\![\mathrm{s}]\!] \overline{\mathrm{w}}'') \wedge (\overline{\mathrm{u}} = \overline{\mathrm{u}}')^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow \bigwedge_{\mathrm{i} \in \mathrm{I}} (\forall \overline{\mathrm{z_i}}.\ \mathrm{P_i}^{\overline{\mathrm{u}}}_{\overline{\mathrm{u}}'} \Rightarrow \mathrm{Q_i})^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''}$$

(for all $\overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''$), which is equivalent to the validity of the conclusion. Thus the validity of the premises for all i is equivalent to the validity of the conclusion. □

Note that the rule may be used backwards, and thereby possibly obtaining triples that were not known before, for instance after reorganizing the postcondition (as illustrated below).

$$\frac{\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\bigwedge_{i \in I} \forall \overline{z_i}.\ P_i{}^{\overline{u}}_{\overline{u}'} \Rightarrow Q_i\}}{\{P_i\}\ \mathrm{s}\ \{Q_i\} \text{ for each } \mathrm{i} \in \mathrm{I}} \quad \text{(BACKWARD GENERALIZED NORMALIZATION)}$$

Example 2 reconsidered. Reconsider the two triples

$\{x \geq 0\}\ \mathrm{s}\ \{x \geq 0\}$ and $\{x \leq 0\}\ \mathrm{s}\ \{x \leq 0\}$.

The generalized normalization rule gives the triple:

$$\{x = x'\}\ \mathrm{s}\ \{(x' \geq 0 \Rightarrow x \geq 0) \wedge (x' \leq 0 \Rightarrow x \leq 0)\}$$

which is a more direct and explicit result than the one obtained above by the generalized adaptation rule. Furthermore, the postcondition can be reformulated as

$$(\mathrm{x}' \geq 0 \Rightarrow \mathrm{x} \geq 0) \wedge (\mathrm{x}' \leq 0 \Rightarrow \mathrm{x} \leq 0) \wedge (\mathrm{x}' = 0 \Rightarrow \mathrm{x} = 0)$$

Backward generalized normalization gives the new triple $\{x = 0\}\ \mathrm{s}\ \{x = 0\}$. This derivation was clearly simpler than with generalized adaptation.

The generalized consequence rule.
By combining the generalized normalization rule with the consequence rule, we obtain a generalized consequence rule

$$\frac{\begin{array}{c}\text{(GENERALIZED CONSEQUENCE)}\\ \{P_i\}\ \mathrm{s}\ \{Q_i\} \text{ for each } \mathrm{i} \in \mathrm{I} \\ (\bigwedge_{\mathrm{i}\in\mathrm{I}} \forall \overline{z_\mathrm{i}} \,.\, \mathrm{P}_{\mathrm{i}\,\overline{u}'}^{\overline{u}} \Rightarrow \mathrm{Q}_\mathrm{i}) \Rightarrow (\forall \overline{z} \,.\, \mathrm{P}_{\overline{u}'}^{\overline{u}} \Rightarrow \mathrm{Q})\end{array}}{\{P\}\ \mathrm{s}\ \{Q\}}$$

(where the free variables in the second premise are implicitly universally quantified). This rule may be used for adaption of known contracts to a specific context. As opposed to all the adaptation rules mentioned above, this rule is agnostic to whether the verification strategy is left-constructive, right-constructive, or top-down (using the terminology of [7]). The generalized normalization rule can directly be used for bottom-up verification.

Example 1 reconsidered. We investigate if we can derive the proposed contracts 1 and 2 assuming x and y are the only program variables. The given contracts are

$(\mathrm{x} < 0,\ \mathrm{x} < 0)$
$(\mathrm{x} > 0,\ \mathrm{x} > 0)$
$(\mathrm{x} = 0,\ \mathrm{x} = 0)$
$(\mathrm{x} < 0 \wedge \mathrm{y} = \mathrm{y}',\ \mathrm{y} = \mathrm{y}')$
$(\mathrm{x} \geq 0 \wedge \mathrm{y} = \mathrm{y}',\ \mathrm{y} = \mathrm{y}' + \mathrm{x})$

According to the generalized consequence rule, we need to show that each of

$$\mathrm{x} < 0 \wedge \mathrm{y} = \mathrm{y}' \vee \mathrm{x} \geq 0 \wedge \mathrm{y} = \mathrm{y}' + \mathrm{x}$$

(from contract 1) and

$$\forall \mathrm{z} \,.\, \mathrm{x}' = 0 \wedge \mathrm{y}' = \mathrm{z} \Rightarrow \mathrm{x} = 0 \wedge \mathrm{y} = \mathrm{z}$$

(from contract 2) follows from the conjunction $(\mathrm{x}' < 0 \Rightarrow \mathrm{x} < 0) \wedge (\mathrm{x}' > 0 \Rightarrow \mathrm{x} > 0) \wedge (\mathrm{x}' = 0 \Rightarrow \mathrm{x} = 0) \wedge (\forall \mathrm{z} \,.\, \mathrm{x}' < 0 \wedge \mathrm{y}' = \mathrm{z} \Rightarrow \mathrm{y} = \mathrm{z}) \wedge (\forall \mathrm{z} \,.\, \mathrm{x}' \geq 0 \wedge \mathrm{y}' = \mathrm{z} \Rightarrow \mathrm{y} = \mathrm{z}+\mathrm{x})$ (reflecting the five given contracts). Note that the logical variable y' appearing in in the contracts becomes quantified according to the second premise of the generalized consequence rule. We rename this quantified variable to z, to adhere to the naming convention of the rule. For the case of contract 1, the implication follows by considering the cases $\mathrm{x}' < 0$ and $\mathrm{x}' \geq 0$. For the case of contract 2, the implication follows by lifting the quantifier on z in the implicand to the outermost level, and instantiating the last quantified z in the implicant (from the fifth contract) to that z.

3 Derivation of General Reasoning Rules

We now consider the reasoning system formed by the *generalized normalization rule* and the *consequence rule*. We illustrate that by these two rules we may quite easily derive common general reasoning rules. We derive below the common rules considered in the classical survey of Apt [1], namely the invariance axiom, substitution rules I and II, conjunction rule, invariance rule, and the elimination rule. According to [1] these rules ensure completeness. We also add some other relevant rules.

3.1 Derivation of Three Instantiation Rules

Assume the triple $\{P\}$ s $\{Q\}$ is given. We may derive $\{\overline{u} = \overline{u}'\}$ s $\{\forall \overline{z}.\ P^{\overline{u}}_{\overline{u}'} \Rightarrow Q\}$ by the normalization rule. By the consequence rule, we obtain $\{\overline{u} = \overline{u}'\}$ s $\{(P^{\overline{u}}_{\overline{u}'})^{\overline{z}}_{\overline{e}} \Rightarrow Q^{\overline{z}}_{\overline{e}}\}$ where $\overline{e}$ is any expression list, possibly referring to program variables. The two substitutions may be merged since $\overline{u}$, $\overline{u}'$, and $\overline{z}$ are disjoint. In the case that Q does not refer to $\overline{z}$, we get $\{P^{\overline{z}}_{\overline{e}}\}$ s $\{Q\}$ by the normalization rule used backwards (which eliminates the substitution on $\overline{u}$). And in the case that the expression list $\overline{e}$ does not refer to program variables that may be changed by s (i.e., $\overline{w}$), the substitution on $\overline{z}$ commutes with the substitution on $\overline{u}$, and we get $\{\overline{u} = \overline{u}'\}$ s $\{(P^{\overline{z}}_{\overline{e}})^{\overline{u}}_{\overline{u}'} \Rightarrow Q^{\overline{z}}_{\overline{e}}\}$, which is equivalent to $\{P^{\overline{z}}_{\overline{e}}\}$ s $\{Q^{\overline{z}}_{\overline{e}}\}$ by the normalization rule used backwards.

Thus we have derived the three rules below (letting $\overline{z}$ denote the logical variables in (P, Q)):

For any expression list $\overline{e}$, including program variables and logical variables, we have

$$\text{(NORMALIZED SUBSTITUTION)}\quad \frac{\{P\}\ \mathrm{s}\ \{Q\}}{\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{P^{\overline{u},\overline{z}}_{\overline{u}',\overline{e}} \Rightarrow Q^{\overline{z}}_{\overline{e}}\}}$$

where the substitutions on $\overline{u}$ and $\overline{z}$ are simultaneous.

For an expression list $\overline{t}$ without program variables, we have

$$\text{(SUBSTITUTION I)}\quad \frac{\{P\}\ \mathrm{s}\ \{Q\}}{\{P^{\overline{z}}_{\overline{t}}\}\ \mathrm{s}\ \{Q^{\overline{z}}_{\overline{t}}\}}$$

Note that program variables $\overline{v}$, not modified by s, may occur in $\overline{t}$.

For the case that $\overline{z}$ does not occur in Q, we have

$$\text{(SUBSTITUTION II)}\quad \frac{\{P\}\ \mathrm{s}\ \{Q\}}{\{P^{\overline{z}}_{\overline{e}}\}\ \mathrm{s}\ \{Q\}}$$

with $\overline{e}$ as above (no restrictions).

3.2 Deriving the Elimination Rule

The *Elimination Rule* states that logical variables in the precondition, not occurring in the postcondition, may be bound by an existential quantifier in the precondition:

$$\frac{\{P\}\ \mathrm{s}\ \{Q\}}{\{\exists \overline{z}.\ P\}\ \mathrm{s}\ \{Q\}}\ (\textsc{elimination})$$

where $\overline{\mathrm{z}}$ are logical variables not occurring in the postcondition Q.

We derive this rule from the normalization rule. The premise is equivalent to $\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\forall \overline{z}, \overline{z}'.\ P^{\overline{u}}_{\overline{u}'} \Rightarrow Q\}$ where $\overline{\mathrm{z}}'$ is the list of logical variables occurring in Q. We may simplify this normalized triple to $\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\forall \overline{z}'.\ (\exists \overline{z}.\ P^{\overline{u}}_{\overline{u}'}) \Rightarrow Q\}$, which again is the same as $\{\exists \overline{z}.\ P\}\ \mathrm{s}\ \{Q\}$, using the normalization rule backwards. Thus we have proved Rule of Elimination, using only the normalization rule.

3.3 Deriving the Invariance Axiom and Rule

The so-called *Invariance Rule* states that one may strengthen both a pre- and postcondition by a predicate R that does not refer to the program variables $\overline{\mathrm{w}}$:

$$\frac{\{P\}\ \mathrm{s}\ \{Q\}}{\{P \wedge R\}\ \mathrm{s}\ \{Q \wedge R\}}\ (\textsc{invariance})$$

Let R be without occurrences of the program variables $\overline{\mathrm{w}}$. We may prove the invariance rule by observing that the premise is (by the normalization rule) equivalent to $\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\forall \overline{z}.\ P^{\overline{u}}_{\overline{u}'} \Rightarrow Q\}$ and that the conclusion is equivalent to $\{\overline{u} = \overline{u}'\}\ \mathrm{s}\ \{\forall \overline{z}, \overline{z}'.\ P^{\overline{u}}_{\overline{u}'} \wedge R \Rightarrow Q \wedge R\}$ where $\overline{\mathrm{z}}'$ are the additional logical variables of R. By Rule of Consequence, it suffices to prove that the former postcondition implies the latter, which is trivial.

Next, we derive the *Invariance Axiom*

$$\{P\}\ \mathrm{s}\ \{P\}$$

for P without occurrences of $\overline{\mathrm{w}}$. Normalization gives $\{true\}\,\mathrm{s}\,\{P \Rightarrow P\}$ since $\mathrm{P}^{\overline{\mathrm{u}}}_{\overline{\mathrm{u}}'}$ reduces to P because P is without occurrences of $\overline{\mathrm{w}}$ and therefore also $\overline{\mathrm{u}}$, and since in this case the precondition $\overline{\mathrm{u}} = \overline{\mathrm{u}}'$ may be replaced by *true*. The rest is trivial, assuming the axiom $\{true\}\ \mathrm{s}\ \{true\}$.

Furthermore, we can derive the *Trivial Axiom*

$$\{true\}\ \mathrm{s}\ \{true\}$$

from any contract (p, q). By consequence we derive $\{false\}\ \mathrm{s}\ \{true\}$ from $\{p\}\ \mathrm{s}\ \{q\}$, and by normalization we obtain $\{true\}\,\mathrm{s}\,\{false \Rightarrow true\}$ since *false* does not refer to any program variables. Then $\{true\}\ \mathrm{s}\ \{true\}$ follows by consequence.

3.4 Deriving the Improved Adaptation Rule

The improved adaptation rule is given by

$$\text{(IMPROVED ADAPTATION)}\quad \frac{\{P\}\ \mathrm{s}\ \{Q\}}{\{\forall \overline{w}'.\ (\forall \overline{z}.\ P \Rightarrow Q^{\overline{w}}_{\overline{w}'}) \Rightarrow R^{\overline{w}}_{\overline{w}'}\}\ \mathrm{s}\ \{R\}}$$

Derivation: The premise is equivalent to $\{\overline{w} = \overline{w}'\}\ \mathrm{s}\ \{\forall \overline{z}.\ P^{\overline{w}}_{\overline{w}'} \Rightarrow Q\}$. The conclusion is equivalent to $\{\overline{w} = \overline{w}'\}\mathrm{s}\{(\forall \overline{w}''.\ (\forall \overline{z}.\ P^{\overline{w}}_{\overline{w}'} \Rightarrow Q^{\overline{w}}_{\overline{w}''}) \Rightarrow R^{\overline{w}}_{\overline{w}''}) \Rightarrow R\}$. By first order logic, the latter postcondition can be reformulated as
$\exists \overline{\mathrm{w}}''.\ ((\forall \overline{\mathrm{z}}.\ \mathrm{P}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow \mathrm{Q}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''}) \Rightarrow \mathrm{R}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''}) \Rightarrow \mathrm{R}$, and it can then be derived from
$((\forall \overline{\mathrm{z}}.\ \mathrm{P}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow \mathrm{Q}) \Rightarrow \mathrm{R}) \Rightarrow \mathrm{R}$, since in general $(\exists \mathrm{w}''.\ \mathrm{A})$ follows from A (where A here is $((\forall \overline{\mathrm{z}}.\ \mathrm{P}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow \mathrm{Q}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''}) \Rightarrow \mathrm{R}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}''}) \Rightarrow \mathrm{R}$). By the consequence rule, this condition follows from $\forall \overline{\mathrm{z}}.\ \mathrm{P}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'} \Rightarrow \mathrm{Q}$, which is the normalized postcondition established from the premise. Since the preconditions $(\overline{\mathrm{w}} = \overline{\mathrm{w}}')$ are the same, we have thus derived the conclusion of the adaptation rule from $\{P\}\ \mathrm{s}\ \{Q\}$.

3.5 Deriving the Generalized Adaptation Rule

For the situation where a number of Hoare triples are known for a program s, we may derive the following adaptation rule, generalized to multiple specifications:

$$\text{(GENERALIZED ADAPTATION)}\quad \frac{\{P_i\}\ \mathrm{s}\ \{Q_i\} \quad \text{for all i}}{\{\forall \overline{w}'.\ (\bigwedge_i \forall \overline{z_i}.\ P_i \Rightarrow Q_i{}^{\overline{w}}_{\overline{w}'}) \Rightarrow R^{\overline{w}}_{\overline{w}'}\}\ \mathrm{s}\ \{R\}}$$

Given the premise $\{P_i\}\ \mathrm{s}\ \{Q_i\}$ for all i, the generalized normalization rule gives
$\{\overline{w} = \overline{w}'\}\ \mathrm{s}\ \{\bigwedge_i \forall \overline{z_i}.\ P_i{}^{\overline{w}}_{\overline{w}'} \Rightarrow Q_i\}$.
Using this result, the derived improved adaptation rule (of Sec. 3.4) gives
$\{\forall w''.\ (\overline{w} = \overline{w}' \Rightarrow \bigwedge_i \forall \overline{z_i}.\ (P_i{}^{\overline{w}}_{\overline{w}'} \Rightarrow Q_i{}^{\overline{w}}_{\overline{w}''})) \Rightarrow R^{\overline{w}}_{\overline{w}''}\}\ \mathrm{s}\ \{R\}$.
Using $\overline{\mathrm{w}} = \overline{\mathrm{w}}'$, we may replace $\mathrm{P_i}{}^{\overline{\mathrm{w}}}_{\overline{\mathrm{w}}'}$ by $\mathrm{P_i}$ and then eliminate $\overline{\mathrm{w}} = \overline{\mathrm{w}}'$ by Rule Substitution II (of Sec. 3.1), replacing $\overline{\mathrm{w}}'$ by $\overline{\mathrm{w}}$. The desired conclusion follows by renaming the quantified variable $\overline{\mathrm{w}}''$ to $\overline{\mathrm{w}}'$.

3.6 Deriving the Conjunction Rule

The *Conjunction Rule* is given by

$$\text{(CONJUNCTION)}\quad \frac{\{P_i\}\ \mathrm{s}\ \{Q_i\} \quad \text{for all i}}{\{\bigwedge_i P_i\}\ \mathrm{s}\ \{\bigwedge_i Q_i\}}$$

As above, the premise gives (by generalized normalization):
$\{\overline{w} = \overline{w}'\}$ s $\{\bigwedge_i \forall \overline{z_i}\,.\, P_i\frac{\overline{w}}{\overline{w}'} \Rightarrow Q_i\}$ which implies $\{\overline{w} = \overline{w}'\}$ s $\{\forall \overline{z}\,.\, (\bigwedge_i P_i\frac{\overline{w}}{\overline{w}'}) \Rightarrow \bigwedge_i Q_i\}$ where $\overline{z}$ is the list of all logical variables, considering all premises. By backward normalization we obtain the desired conclusion.

4 Completeness

We show that the combination of the generalized normalization rule and rule of consequence is relatively complete. Since we only look at general rules, we do not define a specific programming language. Assume that a set of contracts $(\mathrm{P_i},\ \mathrm{Q_i})$ is given for s. If the program text of s is not known, we may not re-verify s to obtain other contracts upon need. Therefore completeness needs to entail that if $\models\{P_i\}$ s $\{Q_i\}$ (for all i) implies $\models\{P\}$ s $\{Q\}$, we should be able to prove $\{P\}$ s $\{Q\}$ from $\{P_i\}$ s $\{Q_i\}$, using only generalized normalization and rule of consequence.

Theorem 1. *(Completeness) Consider a given non-empty set of contracts* $(\mathrm{P_i},\ \mathrm{Q_i})$ *for a statement* s*. If validity of the contracts implies validity of* $\{P\}$ s $\{Q\}$*, then* $\{P\}$ s $\{Q\}$ *can be proved by generalized normalization and consequence, assuming* $\{P_i\}$ s $\{Q_i\}$ *for each* i*.*

Proof. Since the generalized normalization rule can be used both ways, reasoning from several contracts can be reduced to reasoning from one contract. And similarly since the normalization rule can be used both ways, it suffices to consider normalized triples.

It therefore remains to show that if $\models\{\overline{w} = \overline{w}'\}$ s $\{R\}$ implies $\models\{\overline{w} = \overline{w}'\}$ s $\{R'\}$, then we may prove $\{\overline{w} = \overline{w}'\}$ s $\{R'\}$ given a proof of $\{\overline{w} = \overline{w}'\}$ s $\{R\}$, where R is $\bigwedge_{\mathrm{i}}(\forall \overline{\mathrm{z}}\,.\, \mathrm{P_i}\frac{\overline{\mathrm{w}}}{\overline{\mathrm{w}}'} \Rightarrow \mathrm{Q_i})$ and R' is $\forall \overline{\mathrm{z}}\,.\, \mathrm{P}'\frac{\overline{\mathrm{w}}}{\overline{\mathrm{w}}'} \Rightarrow \mathrm{Q}'$. It suffices to show $\mathrm{R} \Rightarrow \mathrm{R}'$ since then we can use the consequence rule to obtain the desired result. By the definition of validity given in Sec. 2.1, we have that
$\forall \overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''\,.\, (\overline{\mathrm{v}}, \overline{\mathrm{w}}'[\![\mathrm{s}]\!]\overline{\mathrm{w}}'') \Rightarrow \mathrm{R}\frac{\overline{\mathrm{w}}}{\overline{\mathrm{w}}''}$ implies $\forall \overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''\,.\, (\overline{\mathrm{v}}, \overline{\mathrm{w}}'[\![\mathrm{s}]\!]\overline{\mathrm{w}}'') \Rightarrow \mathrm{R}'\frac{\overline{\mathrm{w}}}{\overline{\mathrm{w}}''}$.
Consider $\overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''$ such that $\overline{\mathrm{v}}, \overline{\mathrm{w}}'[\![\mathrm{s}]\!]\overline{\mathrm{w}}''$. We then have that $\mathrm{R}\frac{\overline{\mathrm{w}}}{\overline{\mathrm{w}}''}$ implies $\mathrm{R}'\frac{\overline{\mathrm{w}}}{\overline{\mathrm{w}}''}$, thus $\forall \overline{\mathrm{v}}, \overline{\mathrm{w}}', \overline{\mathrm{w}}''\,.\, \mathrm{R}\frac{\overline{\mathrm{w}}}{\overline{\mathrm{w}}''} \Rightarrow \mathrm{R}'\frac{\overline{\mathrm{w}}}{\overline{\mathrm{w}}''}$. We may rename $\overline{\mathrm{w}}''$ to $\overline{\mathrm{w}}$. It follows that R implies R', and thus $\{\overline{w} = \overline{w}'\}$ s $\{R'\}$ follows by rule of consequence from $\{\overline{w} = \overline{w}'\}$ s $\{R\}$, which holds by assumption. □

The theorem may be extended to empty contract sets when adding $\{true\}$ s $\{true\}$ as an axiom. Note that the completeness proof is not depending on a particular programming language, since we only talk about a given (unavailable) program s. We have assumed that the assertion language includes first order logic, but it may not contain the semantic (meta-)relation $[\![\mathrm{s}]\!]$.

For a given programming language and reasoning system that is sound and relatively complete in the sense of Cook [6] when allowing re-verification of programs, we obtain a sound and relatively complete system for reasoning from sets of contracts without re-verification, when adding the generalized normalization rule (and rule of consequence if not already in the system).

5 Related Work

The use of multiple contracts has profound effect on the success of modern software engineering methods supporting various forms of code reuse. Contract-based verification is known though several reasoning frameworks, including proof replay [4], proof reuse [24,14], and proof transformations [26]. Our focus is on contract-based verification, without considering a specific programming language. We therefore focus on general reasoning rules for an arbitrary program. A number of works have considered the problem of adaptation in Hoare-style reasoning and the related soundness and completeness issues. The basic adaptation rules have been discussed in the introduction, and a number of general rules have been discussed in Sec. 3. We refer to Apt and Olderog [1,20] for further overview.

In [23] Pierik and deBoer consider the problem of adaptation for object-oriented programs, based on a closed world assumption. In contrast, the current work is based on an open word assumption, supporting modular reasoning. For object-oriented systems, a behavioral specification of the methods of an interface has the dual role of a contract, on one hand stating implementation requirements to any class implementing the interface, and on the other hand stating properties that classes using objects of the interface may rely on. A well-known problem here is that a behavioral specification of a method of an interface cannot be based on the fields of an implementation as they are not visible in the interface. A possible solution is to use an abstraction of the state, for instance expressed by means of the communication history as in [7], giving rise to completeness[9].

In [5] Bijlsma et al consider contract-based verification of methods, and formulate a rule similar to the improved adaptation rule (Sec. 3.4). They show that the rule is maximally strong, assuming a single contract. They use the example contract $(z \leq x \leq z+1,\ y = z \vee y = z+1)$ to show difficulties with logical variables. Two instantiations of this contract are needed in order to derive the contract $(x = 0,\ y = 0)$. In our case this is possible due to the universal quantifier on $\overline{z}$ in the postcondition of the normalization rule.

A different form of adaptation, based on functional abstraction, is given in [21] and studied in [7]. Functional abstraction assumes that the underlying programming language is deterministic. A deterministic program s is equivalent to **if** t_s **then** $\overline{w} := f_s(\overline{v}, \overline{w})$ **else** abort **fi** for some termination condition t_s and some *effect function* f_s, i.e., a function from the input state to the output state (restricted to $\overline{w}$). Then adaptation reduces to $\{t_s \Rightarrow R^{\overline{w}}_{f_s(\overline{v},\overline{w})}\}\, s\, \{R\}$ where t_s and f_s are symbols, and where the precondition is the weakest possible. Given $\{P\}\, s\, \{Q\}$ we have $t_s \wedge P \Rightarrow Q^{\overline{w}}_{f_s(\overline{v},\overline{w})}$, which provides axiomatic knowledge of the symbols f_s and t_s. For instance, the contract above gives the axiom $t_s \wedge z \leq x \leq z+1 \Rightarrow (f_s(x,y) = z \vee f_s(x,y) = z+1)$. And functional adaptation with postcondition $y = 0$ gives $\{t_s \Rightarrow f_s(x,y) = 0\}\ s\ \{y = 0\}$, which may be reduced to $\{x = 0\}\ s\ \{y = 0\}$, using the axiom and the consequence rule. This approach trivially extends to the case of multiple contracts, letting each contract generate an axiom. However, the approach does not deal with non-deterministic programs.

Reasoning from multiple contracts has been discussed in the setting of modular and incremental reasoning about class inheritance, for the approach of lazy behavioral subtyping [10,11]. Here the set of contracts for a method will in general increase when

moving down in the class hierarchy, which makes it necessary to deal with sets of contracts. A notion of *entailment* and an entailment relation ($\rightarrowtriangle$) were introduced, letting $\{..., (p_i,\ q_i), ...\} \rightarrowtriangle (p,\ q)$ be defined as $(\bigwedge_i \forall \overline{z_i} \,.\, P_i{}^{\overline{w}}_{\overline{w}'} \Rightarrow Q_i) \Rightarrow (\forall \overline{z} \,.\, P^{\overline{w}}_{\overline{w}'} \Rightarrow Q)$ (using our notation). Thus entailment is conceptually similar to the generalized consequence rule. However, the work in [10,11] did not focus on Hoare logic rules, and completeness and soundness of entailment were not discussed.

Our normalization rule is basically expressing the strongest postcondition of a given program, a notion which is well-known in Hoare-style reasoning. Triples expressing the strongest postcondition for preconditions of the form $\overline{w} = \overline{w}'$, known as *most general formulas*, have been used in completeness proofs for specific programming languages [1]. A general rule similar to the normalization rule (called Hoare-SAT) is given by Zwiers et al in [29]. In contrast to our rule, the conclusion is not expressed as a Hoare triple, but rather as a **sat**-specification relating programs and predicates in a complementary formalism. Furthermore, **sat**-specifications are able to deal with multiple specifications. Adaptation can then be made by switching from Hoare logic to **sat**-specifications and back. However, the given *Strong SP adaptation* rule is more complex than our generalized normalization rule since it deals with an arbitrary R, and is similar to the generalized adaptation rule (except from being right-constructive).

Apart from the rules in [29,10,11], there seems to be limited results on Hoare-style reasoning from multiple contracts about the same program (or method). And these rules have rather complex conditions i.e., the precondition is complicated in the rule for left-constructive verification strategy, and the postcondition is complicated in the rule for right-constructive verification. These rules are awkward to use with other strategies – in this case the **sat**-relations would be more flexible. However, the usage of **sat**-relations involves switching between several formal program reasoning systems (Hoare logic and the **sat** system). The generalized normalization rule suggested here is simpler then the ones mentioned and more universally applicable, and it allows adaptation without leaving the setting of Hoare logic.

The reasoning problem considered here is specific to contracts based on pre- and postconditions. As we have seen, there is some flexibility in formulating a pre- and postcondition pair without changing the semantics of the contract. However, the rule of consequence is not insensitive to this flexibility since this rule is based on separate comparison of preconditions and postconditions. Furthermore, the addition of several contracts is not straight forward. In the setting of relational calculus, a contract is basically expressed by a single input/output relation (apart from restrictions of variable sets), and thus the sensitivity to different formulations of pre/postcondition pairs is not an issue. Implication between input/output relations corresponds to semantical entailment. In fact the rule for reasoning about *entailment* of multiple contracts used in lazy behavioral subtyping [10], was derived from relational calculus. And also the **sat** system of [29] is similar to relational calculus. The general rules for relational calculus of programs is simpler than for Hoare logic. Framing of sets of program variables is possible [22]. On the other hand, the notions of pre- and postconditions are useful for specification, and the notions of data invariants, class invariants and loop invariants are expressed more naturally in Hoare logic, and Hoare logic is in general well understood.

The present approach is limited to partial correctness. The normailzation rule is problematic in a total correctness setting, unless a special symbol expressing termination is added (as in [21]). Calculi for contracts based on *refinements* [28,2,17,13], allow reasoning about contracts without mentioning a program, and allow combination of multiple contracts. Furthermore, this setting can deal with both total and partial correctness. Our work is however dedicated to the setting of Hoare-style logics.

6 Conclusions

The present study is triggered by recent work in software engineering methodology, such as software product lines, program evolution, as well as new modularity mechanisms and techniques for dealing with object-orientation and inheritance. The needs of these approaches motivate renewed focus on general reasoning rules, in particular with the added complexity of multiple contracts about the same program. We have not seen a discussion for Hoare Logic on completeness of contract-based verification for the case of multiple contracts. And in this case the importance of simplicity is a concern in itself, since the rules tend to become complex.

We consider partial correctness reasoning and suggest a novel set of general rules, based on generalized normalization (and rule of consequence), and prove soundness and completeness. Normalization is clearly simpler than traditional adaptation and other comparable rules. The generalized normalization rule has only one quantifier, which is a universal quantifier, and no nested implications. The methodology given by comparing normalized postconditions gives a simple and efficient approach to reasoning from sets of contracts, which is also suitable for automatic verification.

Acknowledgment

The authors are indebted to the reviewers for their valuable comments.

References

1. K. R. Apt. Ten years of Hoare's logic: A survey – part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, Oct. 1981.
2. R.-J. Back and M. Butler. Exploring summation and product operators in the refinement calculus. In Proc. *Mathematics of Program Construction*, pages 128–158, Springer 1995.
3. M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: An overview. In *Proc. of the 2004 Intern. Conf. on Construction and Analysis of Safe, Secure, and Interoperable Smart Devices (CASSIS'04)*, vol. 3362 of *LNCS*, pages 49–69. Springer 2004.
4. B. Beckert, R. Hähnle, and P. H. Schmitt. *Verification of Object-oriented Software: The KeY Approach*. Springer 2007.
5. A. Bijlsma, P. A. Matthews, and J. G. Wiltink. A sharp proof rule for procedures in WP semantics. *Acta Informatica*, 26(5):409–419, 1989.
6. S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, Feb. 1978.
7. O.-J. Dahl. *Verifiable Programming*. Intern. Series in Computer Science. Prentice Hall, 1992.

8. F. Damiani, J. Dovland, E. B. Johnsen, O. Owe, I. Schaefer, and I. C. Yu. A transformational proof system for Delta-oriented programming. In *Proc. 16th Intl. Software Product Line Conference, vol. 2 (SPLC'12)*, pages 53–60. ACM 2012.
9. C. C. Din and O. Owe. A sound and complete reasoning system for asynchronous communication with shared futures. *Journal of Logical and Algebraic Methods in Programming*, 83(5-6):360–383, 2014.
10. J. Dovland, E. B. Johnsen, O. Owe, and M. Steffen. Lazy behavioral subtyping. *Journal of Logic and Algebraic Programming*, 79(7):578–607, 2010.
11. J. Dovland, E. B. Johnsen, O. Owe, and M. Steffen. Incremental reasoning with lazy behavioral subtyping for multiple inheritance. *Science of Computer Programming*, 76(10):915–941, 2011.
12. S. Ducasse, O. Nierstrasz, N. Schärli, R. Wuyts, and A. P. Black. Traits: A mechanism for fine-grained reuse. *ACM Transactions on Programming Languages and Systems*, 28(2):331–388, 2006.
13. L. Groves. Refinement and the Z schema calculus. *Electronic Notes in Theoretical Computer Science*, 70(3):70 – 93, 2002. REFINE 2002 (The BCS FACS Refinement Workshop).
14. R. Hähnle, I. Schaefer, and R. Bubel. Reuse in software verification by abstract method calls. In *Automated Deduction – CADE-24: 24th Intern. Conf. on Automated Deduction*, vol. 7898 of *LNCS*, pages 300–314. Springer 2013.
15. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
16. C. A. R. Hoare. Procedures and parameters: An axiomatic approach. In *Symposium on Semantics of Algorithmic Languages*, vol. 188 of *Lecture Notes in Mathematics*, pages 102–116. Springer 1971.
17. B. P. Mahony. The least conjunctive refinement and promotion in the refinement calculus. *Formal Aspects of Computing*, 11(1):75–105, 1999.
18. B. Meyer. Applying "design by contract". *IEEE Computer*, 25(10):40–51, Oct. 1992.
19. B. Meyer. *Eiffel: The Language*. Prentice Hall, 1992.
20. E.-R. Olderog. On the notion of expressiveness and the rule of adaptation. *Theoretical Computer Science*, 24(3):337 – 347, 1983.
21. O. Owe. Notes on partial correctness. Res. Rep. 26, Dept. of Informatics, Univ. Oslo, 1977.
22. O. Owe. On practical application of relational calculus. Res. Rep., Dept. of Informatics, Univ. Oslo, 1992.
23. C. Pierik and F. S. de Boer. Modularity and the rule of adaptation. In *Proc. Algebraic Methodology and Software Technology, AMAST 2004*, pages 394–408. Springer 2004.
24. W. Reif and K. Stenzel. Reuse of proofs in software verification. *Sadhana*, 21(2):229–244, 1996.
25. I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella. Delta-oriented Programming of Software Product Lines. In *SPLC*, vol. 6287 of *LNCS*, pages 77–91. Springer 2010.
26. A. Schairer and D. Hutter. Proof transformations for evolutionary formal software development. In *Algebraic Methodology and Software Technology, AMAST'02*, vol. 2422 of *LNCS*. Springer 2002.
27. N. Schärli, S. Ducasse, O. Nierstrasz, and A. Black. Traits: Composable units of behavior. In *Proc. European Conf. on Object-Oriented Programming (ECOOP)*, vol. 2743 of *LNCS*, pages 248–274. Springer 2003.
28. N. Ward. Adding specification constructors to the refinement calculus. In *FME '93: Industrial-Strength Formal Methods: Formal Methods Europe*. 652–670, Springer 1993.
29. J. Zwiers, U. Hannemann, Y. Lakhnech, F. Stomp, and W.-P. de Roever. Modular completeness: Integrating the reuse of specified software in top-down program development. In *Industrial Benefit and Advances in Formal Methods (FME' 96)*, vol. 1051 of *LNCS*, pages 595–608. Springer 1996.