Process Algebra with Guards

Combining Hoare Logic with Process Algebra

(Extended Abstract)

Jan Friso Groote & Alban Ponse
Department of Software Technology, CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract

We extend process algebra with guards, comparable to the guards in guarded commands or conditions in common programming constructs. The extended language is provided with an operational semantics based on transitions between pairs of a process and a data-state. The data-states are given by a data environment that also defines in which data-states guards hold and how actions (non-deterministically) transform these states. The operational semantics is studied modulo strong bisimulation equivalence. For basic process algebra (without operators for parallelism) we present a small axiom system that is complete with respect to a general class of data environments. In case a data environment \( S \) is known, we add three axioms to this system, which is then again complete, provided weakest preconditions are expressible and \( S \) is sufficiently deterministic.

Then we study process algebra with parallelism and guards. A two phase-calculus is provided that makes it possible to prove identities between parallel processes. Also this calculus is complete. In the last section we show that partial correctness formulas can easily be expressed in this setting and we use process algebra with guards to prove the soundness of Hoare logic for linear processes by translating proofs in Hoare logic into proofs in process algebra.

Note: The first author is supported under ESPRIT Basic Research Action no. 3006 (CON-CUR) and both authors are supported under RACE project no. 1046 (SPECS) but this document does not necessarily reflect the view of the SPECS consortium.

1 Introduction

Hoare logic has been introduced in 1969 as a proof system for the correctness of programs [Hoa69]. Since then it has been applied to many problems, and it has been thoroughly studied (see [Apt81] for an overview). In Hoare logic a program is considered to be a state transformer; the initial state is transformed to a final state. The correctness of a program is expressed by pre- and post-conditions.

More recently, the behaviour of processes has attracted attention. This has led to several process calculi (CCS [Mil80], CSP [Hoa85], ACP [BK84, BW90] and MEIJE [AB84]).
In these calculi the correctness of processes is generally expressed by equations, often saying that a specification and an implementation are equivalent in some sense. The equivalences that are used are mainly based on observations: two processes are equal if some observer cannot distinguish between the two.

It seems a natural and useful question how Hoare logic and process algebra can be integrated. In this paper we provide an answer in two steps. First we extend process algebra with guards. Depending on the state, a guard can either be transparent such that it can be passed, or it can block and prevent subsequent processes from being executed. Typical for our approach is that a guard itself represents a process. With this construct we can easily express the guarded commands of DIJKSTRA [Dij76] and many other conditional constructs in programming and specification languages. The guards in our framework have several nice properties, e.g. they constitute a Boolean algebra. Partial correctness formulas \( \{ \alpha \} p \{ \beta \} \) can be expressed by algebraic equations of the form \( \alpha p = \alpha p \beta \) where \( \alpha \) and \( \beta \) are guards (cf. [MA86]).

We provide process algebra with guards with an operational semantics involving data-state transformations. From now on we use the term 'data-state' to avoid confusion with the notion of a state in process algebra. We consider the processes modulo strong bisimulation equivalence [Mil80] and we come up with two axiomatisations for Basic Process Algebra (without parallelism).

Parallel operators fit easily in the process algebra framework. In Hoare logic, however, parallelism turns out to be rather intricate; proof rules for parallel operators are often substantial [OG76]. In our setup we cannot avoid the difficulties caused by parallel operators in Hoare logic, but we can deal with them in a simple algebraic way.

In the last section of this paper we relate Hoare logic and process algebra. The soundness of a Hoare logic for processes defined by linear recursion [Pon89] can be proved within our algebraic framework.

For proofs omitted here we refer to the full version of this paper [GP90].

Acknowledgement

We thank Jos Baeten, Jan Bergstra, Frank de Boer, Tony Hoare, Catuscia Palamidessi, Fer-Jan de Vries and the referees for their helpful comments.

2 Basic Process Algebra with guards

We extend basic process algebra (BPA [BK84]) with guards. Basic process algebra consists of a given set \( A \) of atomic actions with typical elements \( a, b, \ldots \) and the operators \( + \) (alternative composition) and \( \cdot \) (sequential composition). Atomic actions are viewed as nondeterministic data-state transformers. We assume that we have some set \( G_{at} \) of atomic guards. Guards are viewed as elementary processes that block or are transparent depending on the data-state of the process. We extend the signature \( \Sigma(BPA) \) to the signature \( \Sigma(BPA_{G}) \) by adding basic guards that have as syntax:

\[
\phi ::= \delta \mid \epsilon \mid \neg \phi \mid \psi \in G_{at}
\]

where \( \neg \) is the negation operator on guards, \( \delta \) is the guard that always blocks and \( \epsilon \) is the guard that can always be passed. These last two constants are already well-known in process algebra.