Eliminating the Substitution Axiom from
UNITY Logic

Beverly A. Sanders
Institut für Computersysteme/Departement Informatik, Swiss Federal Institute of Technology
(ETH Zürich), CH-8092 Zürich, Switzerland

Keywords: UNITY; Substitution axiom; Program composition; Completeness

Abstract. The UNITY substitution axiom, “if \((x = y)\) is an invariant of a program, then \(x\) can be replaced by \(y\) in any property of the program”, is problematic for several reasons. In this paper, dual predicate transformers \(sst\) and \(wst\) are introduced that allow the strongest invariant of a program to be expressed, and these are used to give new definitions for the temporal operators \(unless\) and \(ensures\). With the new definitions, the substitution axiom is no longer needed, and can be replaced by a derived rule of inference which is formally justified in the logic. One important advantage is that the effects of the initial conditions on the properties of a program are formally captured in a convenient way, and one can forget about substitution in formal treatments of the UNITY proof system while still having it available when desirable to use during the derivation of programs. Composibility and completeness of the modified logic are also discussed.

1. Introduction

The UNITY programming methodology invented by Chandy and Misra [ChM88] comprises a programming notation and a programming logic. A program consists of variable declarations, initial conditions, and a finite set of multiple, conditional assignment statements. The programming logic is based on the temporal operators \(unless\), \(ensures\), and \(\rightarrow\) (read leads-to), plus \(invariant\) and the substitution axiom. Although both the language and the logic...
are surprisingly simple, Chandy and Misra and others have demonstrated with a large number of examples that this method is a tractable way to formally derive a wide variety of interesting parallel algorithms. UNITY promises to become an increasingly important tool in the future.

Probably due to the informal way it is stated, together with the very reasonable sounding justification for it as a generalisation of Leibniz's Rule, the substitution axiom has been widely neglected and misunderstood. Also, due to confusing notation in [ChM88], many have misunderstood the definitions of the temporal operators unless and ensures, believing that the definitions are intended to be equivalence relations rather than inference rules. As will be explained later, combining the substitution axiom with the incorrectly defined temporal operators gives an unsound proof system. If the substitution axiom is omitted, then the incorrect definitions (which do have desirable properties) can be used and the proof system is sound, but incomplete (i.e. there are properties that can be proved with the substitution axiom but not without it). Most of the theoretical studies of UNITY of which I am aware [GeP89, JKR89, Kna89, Liu89] have used the incorrect definitions without the substitution axiom. In [Go190], the incorrect definitions of unless and ensures are used and substitution can only be applied to $\leftrightarrow$ properties. Misra has clarified the definitions in [Mis90b], but we are still left with an informally stated substitution axiom which is difficult to handle and whose role in the logic is obscure. Also, theorems about program composition are difficult to express.

In this paper, modifications to UNITY logic are proposed that eliminate the need for the substitution axiom and also have the desirable properties of the "incorrect" definitions of unless and ensures. The modified logic is also relatively complete. The important concept is the strongest invariant of a program which can be expressed using dual predicate transformers $sst$ and $wst$, defined below. In addition, subscripted properties are defined and used in new theorems for program composition.

2. A Short Introduction to UNITY

This section presents the most essential aspects of UNITY. For more information, and many examples using it to derive programs, see Chandy and Misra's book [ChM88].

2.1. Programing Notation and Operational Interpretation

For our purposes, a UNITY program $F$ consists of three sections. The declare section, $F.DECLARE$ is a set of variable declarations (variable names together with their types), the initially section, $F.INIT$, is a predicate that characterises the allowed initial states, and the assign section, $F.ASSIGN$, is a non-empty set of multiple, conditional assignment statements. The always section is not considered in this paper since any program with a non-empty always section can be transformed to an equivalent program without one. Where no confusion can arise, the name of the program will often be omitted. An execution of a UNITY program would begin in a state satisfying INIT and repeatedly execute (atomically) statements in the assign section. The choice of