

# Sometime = Always + Recursion $\equiv$ Always on the Equivalence of the Intermittent and Invariant Assertions Methods for Proving Inevitability Properties of Programs

Patrick Cousot<sup>1</sup> and Radhia Cousot<sup>2</sup>

<sup>1</sup> Ecole Polytechnique, Centre de Mathématiques Appliquées, F-91128 Palaiseau Cedex, France

<sup>2</sup> Université de Paris-Sud, Centre d'Orsay, LRI, Bat. 490, F-91405 Orsay Cedex, France

**Summary.** We propose and compare two induction principles called "always" and "sometime" for proving inevitability properties of programs. They are respective formalizations and generalizations of Floyd invariant assertions and Burstall intermittent assertions methods for proving total correctness of sequential programs whose methodological advantages or disadvantages have been discussed in a number of previous papers. Both principles are formalized in the abstract setting of arbitrary nondeterministic transition systems and illustrated by appropriate examples. The "sometime" method is interpreted as a recursive application of the "always" method. Hence "always" can be considered as a special case of "sometime". These proof methods are strongly equivalent in the sense that a proof by one induction principle can be rewritten into a proof by the other one. The first two theorems of the paper show that an invariant for the "always" method can be translated into an invariant for the "sometime" method even if every recursive application of the later is required to be of finite length. The third and main theorem of the paper shows how to translate an invariant for the "sometime" method into an invariant for the "always" method. It is emphasized that this translation technique follows the idea of transforming recursive programs into iterative ones. Of course, a general translation technique does not imply that the original "sometime" invariant and the resulting "always" invariant are equally understandable. This is illustrated by an example.

## 1. Introduction

We compare two induction principles (that we call "always" and "sometime") for proving inevitability properties of transition systems, the nondeterminism of which can be unbounded. These induction principles are formalizations of program proof methods independently of a particular programming language, of a particular inevitability property and of a particular style of presentation

of proof methods. Thanks to these abstract and concise formalizations we can make more rigorous comparisons of program proof methods which for some time have always been compared using methodological arguments based on examples [8, 12].

We introduce the induction principle called "always" in [4]. It is a generalization of those program proof methods in the class of Floyd [7] invariant assertions method for proving total correctness of sequential programs. We introduce the induction principle called "sometime" in [5]. It is an abstract generalization of Burstall [2] intermittent assertions method for proving total correctness of sequential programs.

The "always" induction principle involves an induction along execution traces whereas the "sometime" induction principle involves a combination of an induction along (parts of) execution traces (connected to Burstall's "hand simulation") and recursion induction (related to Burstall's "induction on the data"). Hence the "always" induction principle corresponds to the particular case of the "sometime" induction principle when recursion is not used.

The "always" and "sometime" induction principles are sound and semantically complete (see the proofs respectively in [4, 5]) hence weakly equivalent in the sense that when a proof exists by one method, a proof must exist by the other method. Here we prove a stronger equivalence result, in the sense that whenever a proof exists by one method it can be rewritten into a proof by the other method. Intuitively this is because recursion can be eliminated from proofs in favour of induction in just the same way as it can be eliminated from programs in favour of iteration.

We illustrate our techniques by examples.

## 2. Inevitability Properties of Programs Represented as Transition Systems

To introduce abstract formalizations of program proof methods, we view programs as (nondeterministic) transition systems  $\langle S, t \rangle$  [9] where  $S$  is a nonempty set of states and the transition relation  $t$  between a state and its possible successors is represented as a function  $t \in (S \times S \rightarrow \{tt, ff\})$  from pairs of states into truth values ( $tt$  is true and  $ff$  is false).

We say that the nondeterminism of  $t$  is *bounded* when states can only have a finite number of successor states that is to say when  $\forall s \in S. |\{s' \in S: t(s, s')\}| \in \omega$  (where  $|E|$  is the cardinal of a class  $E$  and  $\omega$  is the set of natural numbers) and *unbounded* otherwise. There is no need for the nondeterminism of  $t$  to be bounded. Hence we can represent parallel programs and even fair ones as transition systems by decomposition of these programs into their atomic actions. This decomposition usually destroys syntactic structure of programs and is therefore a debatable way of proceeding. However we neither suggest that program proofs should make a direct use of our induction principles nor that programs should be represented as transition systems before initiating the proofs. On the contrary we attach a great importance to the presentation of proofs. In [3] we have proposed a systematic method for constructing proof methods from induction principles, so that, in particular, the structural information about

