On the completeness of a proof system for a simple data-parallel programming language (extended abstract)*

Luc Bougé**, David Cachera

LIP, ENS Lyon, 46 Allée d’Italie, F-69364 Lyon Cédex 07, France

Abstract. We prove the completeness of an assertional proof system for a simple loop-free data-parallel language. This proof system is based on two-part assertions, where the predicate on the current value of variables is separated from the specification of the current extent of parallelism. The proof is based on a Weakest Precondition (WP) calculus. In contrast with the case of usual scalar languages, not all WP can be defined by an assertion. Yet, partial definability suffices to prove the completeness thanks to the introduction of hidden variables in assertions. The case of data-parallel programs with loops is briefly discussed in the conclusion.

Keywords: Concurrent Programming; Specifying and Verifying and Reasoning about Programs; Semantics of Programming Languages; Data-Parallel Languages; Proof System; Hoare Logic; Weakest Preconditions.

1 Introduction

The development of massively parallel computing in the last two decades has called for the elaboration of a parallel programming model. The data-parallel programming model has proven to be a good framework, since it allows the easy development of applications portable across a wide variety of parallel architectures. The increasing role of this model requires appropriate theoretical foundations. These foundations are crucial to design safe and optimized compilers, and programming environments including parallelizing, data-distributing and debugging tools. They are also the way to safe programming techniques, so as to avoid the common waste of time and money spent in debugging.

Existing data-parallel languages, such as HPF, C*, HYPERC or MPL, include a similar core of data-parallel control structures. In previous papers, we have shown that it is possible to define a simple but representative data-parallel kernel language (the $L$ language), to give it a formal operational [5] and denotational semantics [4], and to define a proof system for this language, in the style of the usual Hoare’s logic approach [10, 4]. The originality of our approach lies in the treatment of the extent of parallelism, that is, the subset of currently active

---

* The full version of this paper can be found in [2].
** Authors contact: Luc Bougé (Luc.Bouge@lip.ens-lyon.fr). This work has been partly supported by the French CNRS Research Program on Parallelism, Networks and Systems (PRS).
indices at which a vector instruction is to be applied. Previous approaches led to manipulate lists of indices explicitly [6, 11], or to consider context expressions as assertions modifiers [8]. In contrast, our proof system for $\mathcal{L}$ describes the activity context by a vector boolean expression distinct from the usual predicates on program variables.

We have shown that our proof system for $\mathcal{L}$ is sound, that is, any provable property of a program is actually valid. In this paper, we address the converse completeness problem: Can any valid property of a program be proved in our system? In some sense, completeness guarantees that the rules of a proof system actually catch all the semantic expressiveness of the language under study.

The completeness of proof systems for scalar Pascal-like languages has been extensively studied [1]. In attacking such a problem, the main tool is the weakest preconditions calculus. This notion has been introduced by Dijkstra [7]. It plays a central role in the formal validation of scalar programs, as shown in [9] for instance. The case of data-parallel programs is much more complex than the case of scalar programs, as one has to cope both with the variable values and with the manipulations of the activity context. Yet, we have shown in [3] that it is possible to define a weakest preconditions calculus for $\mathcal{L}$, at least for loop-free (so-called straight-line or linear) programs.

The contribution of this paper is to apply these results to prove the completeness of our proof system for all linear programs. We proceed as follows. We first present the $\mathcal{L}$ language, and give its denotational semantics. We describe a sound assertional proof system for this language, as defined in [4], and its weakest preconditions calculus as described in [3]. Then, we prove the completeness of the proof system in a restricted case: plain specifications formulae and regular programs. To handle non-regular programs, we extend the proof system with an additional rule. It enables to introduce and eliminate auxiliary hidden variables in assertions. We prove that this extended proof system is complete for linear programs without any restriction.

2 The $\mathcal{L}$ Language

An extensive presentation of the $\mathcal{L}$ language can be found in [5]. For the sake of completeness (if we dare say so!), we briefly recall its denotational semantics as described in [3].

2.1 Informal Description

In the data-parallel programming model, the basic objects are arrays with parallel access. Two kinds of actions can be applied to these objects: component-wise operations, or global rearrangements. A program is a sequential composition of such actions. Each action is associated with the set of array indices at which it is applied. An index at which an action is applied is said to be active. Other indices are said to be idle. The set of active indices is called the activity context. It can be seen as a boolean array where true denotes activity and false idleness.

The $\mathcal{L}$ language is designed as a common kernel of data-parallel languages like C*, HYPERC or MPL. We do not consider the scalar part of these languages, mainly imported from the C language. For the sake of simplicity, we consider a