Event-B Decomposition for Parallel Programs

Thai Son Hoang and Jean-Raymond Abrial

Department of Computer Science,
Swiss Federal Institute of Technology Zurich (ETH-Zurich),
CH-8092, Zurich, Switzerland
htson@inf.ethz.ch, jrabrial@neuf.fr

Abstract. We present here a case study developing a parallel program. The approach that we use combines refinement and decomposition techniques. This involves in the first step to abstractly specify the aim of the program, then subsequently introduce shared information between sub-processes via refinement. Afterwards, decomposition is applied to split the resulting model into sub-models for different processes. These sub-models are later independently developed using refinement. Our approach aids the understanding of parallel programs and reduces the complexity in their proofs of correctness.

Keywords: Event-B, parallel programs, decomposition, refinement.

1 Introduction

We consider here programs that use several co-operating parallel processes in order to compute the intended final result. Proving correctness of such programs is a difficult task because of the interleaved execution of many sub-statements from different processes. These sub-statements may be executed in an unpredictable order. As a result, techniques such as program testing do not give us sufficient confidence about the correctness of these programs, since no execution leading to an error might appear during tests. To achieve correctness, it is therefore necessary to develop these programs and prove them formally.

There are a number of methods for proving the correctness of parallel programs [11]. Our main contribution is an approach applying the technique of refinement and decomposition in Event-B [2], which reduce the complexity of the verification process (more information in Section 5.1). The approach contains four steps as follows.

1. Start with an abstract specification in-one-shot giving the purpose of the program.
2. Refine this abstract specification by introducing details about the shared variables.
3. Decompose the model in the previous step to split the model into several (abstract) sub-models for processes.
4. Refine each sub-model from the previous step independently.

In the last step, each sub-model can be seen as a new abstract specification and hence application of steps 2, 3 and 4 can be repeated again. The novelty of our approach is in

* Part of this research was carried out within the European Commission ICT project 214158 DEPLOY (http://www.deploy-project.eu/index.html). We thank Matthias Schmalz, Christoph Sprenger and David Basin for their comments on drafts of this paper.
step 2 where we specify shared information between processes. This information has two purposes. Firstly, it contains the necessary guarantee condition from each process to establish the final result. Secondly, it also gives the condition on which each process can rely on in further development. This decision to have this step early in our development takes advantage of our decomposition technique and results in simpler models and reduces the complexity of proving programs. This is the main advantage of our method over existing approaches. More information on related work is in Section 5.1.

The rest of the paper is structured as follows. Section 2 gives an overview of the Event-B method and the concept of (shared variable) decomposition. Section 3 introduces the FindP program and its formal development using our approach is presented in Section 4. Section 5 compares our approach with some existing methods for developing parallel programs and draws some conclusions.

2 The Event-B Modelling Method

A development in Event-B [6] is a set of formal models. The models are built from expressions in a mathematical language, which are stored in a repository. When presenting our models, we will do so in a pretty-printed form e.g., adding keywords and following layout conventions to aid parsing. Event-B has a semantics based on transition systems and simulation between such systems, described in [3]. We will not describe in detail the Event-B semantics here and instead just illustrate some of the proof obligations that are important for our development.

Event-B models are organised in terms of the two basic constructs: contexts and machines. Contexts specify the static part of a model whereas machines specify the dynamic part. Contexts may contain carrier sets, constants, axioms, and theorems. Carrier sets are similar to types [6]. Axioms constrain carrier sets and constants, whereas theorems express properties derivable from axioms. In the following, we further describe machines and machine refinement.

2.1 Machines

Machines specify behavioural properties of Event-B models. Machines may contain variables, invariants, theorems, events, and variants. Variables v define the state of a machine. They are constrained by invariants I(v). Possible state changes are described by events. Each event is composed of a guard G(t, v) (the conjunction of one or more predicates) and an action S(t, v), where t are the parameters of the event[1] The guard states the necessary condition under which an event may occur, and the action describes how the state variables evolve when the event occurs. An event can be represented by the term “any t where G(t, v) then S(t, v) end”. We use the short form “when G(v) then S(v) end” when the event does not have any parameters, and we write “begin S(v) end” when, in addition, the event’s guard equals true. A dedicated event of the last form is used for initialisation.

[1] When referring to variables v and parameters t, we usually allow for multiple variables and parameters, i.e., they may be “vectors”.