ARTICLE IN PRESS

Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Playing with state-based models for designing better algorithms

Dominique Méry

Université de Lorraine, LORIA, BP 239, 54506 Vandœuvre-lès-Nancy, France

HIGHLIGHTS

- We develop a translation of annotated programs into Event-B models.
- We extend the scope of translation for concurrent programs.
- We develop a combination of the refinement of Event-B models and the coordination paradigm.
- We illustrate techniques on case studies.

ARTICLE INFO

Article history:
Received 6 May 2015
Received in revised form
6 March 2016
Accepted 30 April 2016
Available online xxxx

Keywords:
Modeling languages
Verification
Refinement
Coordination
Algorithm

ABSTRACT

State-based models provide a very convenient framework for analyzing, verifying, validating and designing sequential as well as concurrent or distributed algorithms. Each state-based model is considered as an abstraction, which is more or less close to the target *algorithmic* entity. The problem is then to organize the relationship between an initial abstract state-based model expressing requirements and a final concrete state-based model expressing a structured *algorithmic* state-based model. A *simulation* (or *refinement*) relation between two state-based models allows to structure these models from an abstract view to a concrete view. Moreover, state-based models can be extended by assertion languages for expressing correctness properties as *pre/post specification*, *safety* properties or even *temporal properties*. In this work, we review state-based models and *play scores* for verifying and designing concurrent or distributed algorithms. We choose the Event-B modeling language for expressing state-based models and we show how we can play *Event-B scores* using Rodin and methodological elements to guarantee that the resulting algorithm is correct with respect to initial requirements. First, we show how annotation-based verification can be handled in the Event-B modeling language and we propose an extension to handle the verification of concurrent programs. In a second step, we show how important is the concept of refinement and how it can be used to found a methodology for designing concurrent programs using the coordination paradigm.

1. Introduction

State-based models play a central role in the verification, validation and design of software-based systems. They provide a very convenient and flexible framework for developing both techniques and tools to improve quality of software-based systems. We review seminal results on proving the correctness of algorithms and we use the Event-B modeling language with the Rodin environment for illustrating our contributions.

First, we review the classical annotation-based method [1–4], known as Floyd–Hoare's method, and report an experience with MSc students for checking the correctness of sequential and concurrent programs by transformation of annotated algorithms into state-based models in the Rodin environment. The transformation of annotated programs into state-based models is a very general transformation with respect to safety properties; it allows to state verification conditions as proof obligations of the Event-B resulting machine and to obtain, for free, a platform for deductive program verification. This first step shows that the design of correct-by-construction parallel algorithms remains a tricky task especially in the annotation process which is supposed to discover invariants of concurrent programs. State-based models help *concurrent and parallel programs to meet provers* [5] and the Event-B modeling language is a rich and extensible set-theoretical language for handling properties as partial correctness and absence of run-time errors. The Rodin platform offers an interactive proof assistant and SMT solvers, which are discharging majority of

E-mail address: dominique.mery@loria.fr.
URL: http://www.loria.fr.

http://dx.doi.org/10.1016/j.future.2016.04.019
0167-739X/

Please cite this article in press as: D. Méry, Playing with state-based models for designing better algorithms, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.04.019

generated proof obligations. The remaining non-discharged proof obligations require a specific interactive processing either for modifying the program or for modifying the annotations. The possible remaining non-discharged proof obligations demand a specific care of the user and this observation leads us to reconsider the refinement in the design of parallel programs using informations of the design and to reformulate the PCAM methodology based on state-based models for addressing practical concurrent and parallel programs [6,7].

Second, from the previous experience, we illustrate how the refinement can improve and facilitate the verification process by relating state-based models. We describe a simple extension of the *call-as-event* paradigm [8,9] to handle the design of concurrent programs in the *coordination* [10]-based approach. More precisely, I. Foster [6] has introduced the PCAM methodology for designing concurrent programs and we combine the refinement and the coordination paradigm for deriving state-based models identified in the design of concurrent programs following the PCAM approach. PCAM is a design methodology for parallel programs and starting with a problem specification, it develops a Partition, determines Communication requirements, Agglomerate tasks, and finally Maps tasks to processors: PCAM stands for Partition, Communication, Aggregation, Mapping. These four transformations can be interpreted as refinement step preserving safety properties of state-based models, which communicate using the *coordination* paradigm. Refinement allows to progressively structure state-based models with the help of the coordination technique and to enrich the current invariant.

The refinement-based or stepwise development of algorithms has been first initiated in the seminal works of Dijkstra [11], Back [12] or Morgan [13]. Next, UNITY [14] has proposed a rich framework for designing distributed algorithms combining a simple temporal logic for expressing required properties and a simple language for expressing actions modifying state variables under fairness assumption. TLA/TLA$^+$ [15][1] proposes a general modeling language based on a temporal logic of actions combined with a set-theoretical modeling language for data and is extended by a specific algorithmic language namely PlusCal [16],[2] which is translated into TLA$^+$ and which is closer to the classical way to express a distributed algorithm.

More recently, Event-B [17,18] provides a technique for incremental and proof-based development of reactive systems and is supported by open tools [19,20]. It integrates set-theoretical notations, a first-order predicate calculus and structures called machines as models; it includes the concept of models refinement expressing that, if a machine M refines a machine N, M can only behave in a way that corresponds to the behavior of M. An Event-B machine models a reactive system i.e. a system driven by its environment and reacting to its stimuli. An important property of these machines is that its events preserve the invariant properties defining a set of reachable states. The Event-B method has been developed from the classical B method [21] and it offers a general framework for developing the *correct-by-construction* systems by using an incremental approach for designing the models by refinement. Refinement [11,12] is a relationship relating two models such that one model is refining or simulating the other one. When an abstract model is refined by a concrete model, it means that the concrete model simulates the abstract model and that any safety property of the abstract model is also a safety property of the concrete model. In particular, the concrete model preserves the invariant properties of the abstract model.

UNITY [14] contributes to the design of concurrent programs by combining actions systems, temporal logics and refinement. The contribution of UNITY is not only technical but also methodological. The action system language proposes a very simple way to express computations and it emphasizes on the role of the abstraction in the design of concurrent programs: the development should start by a set of actions modeling the target concurrent system without the complex syntactical entities involved in the concurrent programming. The choice is to use a simple programming language and to postpone the real programming language in the mapping phase. The main problem of UNITY is that it remains a formal approach without effective environment for assisting the user; the UNITY modeling language is too rich and includes liveness properties as well as fairness assumption. *Grasp all, lose all* would be a sentence for warning users to use a too complex modeling language and to have to manage too many features when progressing. Our solution is to restrict the modeling language by considering safety properties and by addressing issues on termination as proposed by Abrial, using the convergent events of Event-B. Our target applications require a simple expression of fairness. Further improvements are possible as M. Poppleton and myself [22] have demonstrated for population protocols.

The Gamma [23] model develops the chemical programming [24] and the rule-based programming; it allows a simple design of complex problems and is based on the use of multisets as working data structures. Gamma and UNITY are based on simple programming constructs to help the designer to focus on the correctness of the developed programs and to delegate questions on the parallel architecture in final steps of development. Gamma aims to develop a methodology for simplifying the construction of concurrent programs but is not proposing an incremental process for deriving the final model as well as the invariant. Blass and Gurevich [25] propose a formulation of parallel algorithms in the ASM framework and introduce concepts as *ken* and *proclets* with postulates for proving the parallel thesis. Their work is related to the parallel computing model rather than to the parallel programming model, which is our objective.

In the same direction, I. Foster [6] introduces the PCAM methodology which is based on the coordination paradigm and a decomposition of the stages for designing a concurrent program. The objective is to produce real concurrent programs in a real programming language from a systematic analysis of the problem. The two first stages of PCAM concern the identification of the set of tasks and the communications among tasks; they can be made more systematic and can be expressed using a combination of the coordination paradigm and the refinement in Event-B. It is why we are stating that we play with state-based models using patterns for deriving concurrent programs. These patterns are called archetypes [26] by Massingill and Chandy. These different approaches show the importance of separating concerns and to develop specific proof-based patterns combining the refinement and another paradigm as, for instance, we did with the call-as-event approach [8,9], combining the refinement and the recursivity or the service-as-event paradigm combining the refinement and the temporal logic [27]. In the two last experiences, we have facilitated the discovery of invariants and the proof process.

Having motivated our objectives, we organize the paper as follows. Section 2 is introducing the Event-B modeling language. Section 3 develops a systematic translation of annotated sequential algorithm into an Event-B model for checking the correctness of the annotation and for deriving partial correctness of the annotated algorithm. Section 4 is an extension of the transformation to concurrent programs. Section 5 is using the refinement together with the coordination paradigm for designing concurrent programs. Section 6 presents conclusion and future works.

[1] TLA stands for Temporal Logic of Actions and TLA$^+$ stands for TLA+.

[2] PlusCAL (formerly called +CAL) is an algorithm language based on TLA$^+$ and a PlusCal algorithm is translated to a TLA$^+$ specification.

Please cite this article in press as: D. Méry, Playing with state-based models for designing better algorithms, Future Generation Computer Systems (2016), http://dx.doi.org/10.1016/j.future.2016.04.019