Science of Computer Programming 92 (2014) 105–128

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Compositionality and correctness of fault tolerant patterns in HOL4

Diego Dias, Juliano Iyoda *

Center of Informatics, Federal University of Pernambuco, Recife, CEP 50.740-560, Brazil

ARTICLE INFO

Article history:
Received 8 April 2012
Received in revised form 13 July 2013
Accepted 17 July 2013
Available online 1 August 2013

Keywords:
Redundancy management
Fault tolerance
Behavioural preservation
Theorem proving
HOL

ABSTRACT

In the development of critical systems, it is common practice to make use of redundancy in order to achieve higher levels of reliability. There are well established design patterns that introduce redundancy and that are widely documented and adopted by the industry. However there have been few attempts to formally verify them. In this work, we modelled in the HOL4 system such design patterns, which we call here *fault tolerant patterns*. We illustrate our approach by modelling three classical fault tolerant patterns: Homogeneous Redundancy, Heterogeneous Redundancy and Triple Modular Redundancy. Our model takes into account that the original system (without redundancy) computes a certain function with some delay and is amenable to random failures.

We proved that our fault tolerant patterns preserve the behaviour of its replicated subsystems. The notion of correctness adopted makes use of interval arithmetic and is restricted to functional behaviour. Timing is not regarded as part of the functional behaviour in this work. Therefore, real-time systems are not the focus of our approach. We also proved that our fault tolerant patterns are compositional in the sense that we can apply fault tolerant patterns consecutively and for an arbitrary number of times. The consecutive application of our patterns still results in a system that computes a certain function with some delay and amenable to random failures. We developed a case study that verifies that a fault tolerant pattern applied to a simplified avionic Elevator Control System preserves its original behaviour. This work was done in collaboration with the Brazilian aircraft manufacturer Embraer.

1. Introduction

Critical systems are developed in such a way that safety is addressed explicitly and under severe regulations. For instance, catastrophic failure of digital flight control systems must be extremely improbable to occur: the failure rate must be lower than 10^{-9} per hour [1]. In order to satisfy such requirement, critical systems are often replicated in different ways in order to guarantee that a failure on one component is covered by another replica of it [2]. Therefore fault tolerance is mostly based on redundancy.

Redundancy is implemented in different ways. We can replicate a system with an identical copy of it (and add a monitor to check whether at least one of them is working); or we can use replicas of a system that have different design and implementation, but that computes the same function; or we can ask for a voter to output an average value of the output of the replicas; and so on. These design solutions, which we call here *fault tolerant patterns*, are widely used in industry.

* Corresponding author.
E-mail addresses: dmd@cin.ufpe.br (D. Dias), jmi@cin.ufpe.br (J. Iyoda).

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.07.009

Fault tolerant patterns are *de facto* standards in industry. Unfortunately, there has been few attempts to formally verify this catalogue of design patterns. In this paper, we model in HOL4 [3] three fault tolerant patterns widely used in industry, namely, Homogeneous Redundancy, Heterogeneous Redundancy and Triple Modular Redundancy. Our specification models the original (non-replicated) system as a black-box that computes a certain function with some delay and is amenable to random failures. We proved two sets of theorems. One set establishes that our fault tolerant patterns are a system that computes a certain function with some delay and is amenable to random failures. These theorems capture the fact that our fault tolerant patterns are compositional: we can apply the patterns an arbitrary number of times and the result is still such a system whose interface is the same of the original non-replicated system. The second set of theorems establishes the preservation of the behaviour of the fault tolerant patterns with respect to its subsystems. Our notion of correctness makes use of interval arithmetic: our fault tolerant pattern never outputs a value outside the combined range of the output of its subsystems. This paper extends our work published previously [4] in which behavioural preservation was justified informally. In this paper we define preservation of behaviour of a composite system as the property of producing outputs that lay in the same range of the output of its subsystems. Differently from our previous work [4], this new notion of behavioural preservation has been fully formalised and all correctness theorems have been proved in the HOL4 system (see Section 4.2). Timing is not regarded as functional behaviour in our approach. Therefore, real-time systems are not the focus of this work.

Pioneering work on the verification of fault tolerant patterns was done in the nineties [5–7]. These works proved the correctness of fault tolerant patterns by hand [5], by using theorems provers [6], and with model checkers [7]. More recently, Dajani-Brown et al. [8] used SCADE's [9] model checker to verify the correctness of a triple redundant system. Our work differs from previous works on the compositionality of our theorems and on the separation of concerns: the failure rate and the functional behaviour of the system are separate entities. We do not assume any specific failure rate of the system in order to prove the behavioural preservation (contrary to previous works). Compositionality has been addressed by Cofer et al. [10] using contracts of a system that are derived from the contracts of its subsystems.

This work is concerned with the non-introduction of design error by redundancy. We are not concerned with the benefits of redundancy with respect to failure rate improvements. There are other studies [11,12] that show the quantitative benefits of using redundancy and discuss aspects as cost increase and modifiability analysis.

We illustrate our approach with a case study in which we apply the Triple Modular Redundancy pattern to a simplified model of an avionics elevator. We show that it is easy to prove that the addition of the redundancy did not introduce more faults to the original (non-replicated) system. This work has been done in collaboration with the Brazilian aircraft manufacturer Embraer.

This paper is organised as follows. Section 2 introduces the background for this work. We briefly introduce higher order logic in HOL4 and describe how it can be used to model hardware. We also introduce the architectures of the fault tolerant patterns and how they work. Section 3 presents our model in HOL4 of three fault tolerant patterns. Section 4 shows the compositionality and the correctness theorems of the fault tolerant patterns. Section 5 illustrates our approach in a case study. Section 6 presents the related work in more detail and Section 7 concludes.

2. Background

This section presents the background for our work. We start by introducing higher order logic and one of its implementation: the HOL4 system. We also show how higher order logic can be used to model hardware. Our fault tolerant patterns are modelled following the same principles used in hardware. Finally, we introduce the three fault tolerant patterns we formalised: Homogeneous Redundancy, Heterogeneous Redundancy and Triple Modular Redundancy.

2.1. Higher order logic

HOL stands for Higher Order Logic and, in this work, it also refers to the particular formulation developed by Mike Gordon at the University of Cambridge in the 1980s [13]. HOL is a predicate calculus with terms from the typed lambda calculus [14]. The notation used to represent the logical operations in HOL is similar to the conventional predicate calculus.

There are only four kinds of terms in HOL: constants, variables, applications and λ-abstractions. A lambda abstraction $\lambda x.P(x)$ denotes an anonymous function that maps its argument x to the value $P(x)$. Variables can range over functions (λ-abstractions) and predicates. Functions can take functions as arguments and return functions as results. As in λ-calculus, application has the form $(M\ N)$ and denotes the result of applying the function M to the value N. By convention, application is left-associative: $f\ x_1\ x_2\ \ldots\ x_n = (((f x_1)x_2)\ldots x_n)$. For convenience, there are many syntactic sugars in HOL. For instance, the logical constant *true* is defined as $\mathsf{T} = ((\lambda x.x) = (\lambda x.x))$, while "for all" is $\forall = (\lambda P.P = \lambda x.T)$, and so on.

Table 1 summarises a subset of the (sugared) notation of HOL with a short description for each term. These terms represent the constants *true* and *false*, logical operations (negation, disjunction, conjunction and implication), variable quantification and a conditional operation. From now on we will use the syntax presented in Table 1 instead of HOL's primitive definitions.

Every term is associated with a unique type, which denotes a set. There are four kinds of types in the HOL logic described by the following BNF grammar:

$$\mathsf{T} ::= \alpha \quad | \quad c \quad | \quad (\mathsf{T}_1, \ldots, \mathsf{T}_n)\ op \quad | \quad \mathsf{T}_1 \rightarrow \mathsf{T}_2$$