National Institute of
Advanced Industrial Science
and Technology
AIST

AIST-PS-2010-007

A note on "D-Cases as proofs as programs"

Makoto Takeyama

Collaborative Research Team for Verification and Specification
National Institute of Advanced Industrial Science and Technology

算譜科学研究速報
Programming Science
Technical Report

独立行政法人 産業技術総合研究所 組込みシステム技術連携研究体 発行
Published by Collaborative Research Team for Verification and Specification
National Institute of Advanced Industrial Science and Technology

A note on "D-Cases as proofs as programs"

Maktoto Takeyama *
Collaborative Research Team for Verification and Specification
National Institute of Advanced Industrial Science and Technology

October 28, 2010

Abstract

A D-Case, a structured argument for Open System Dependability, can be seen as a formal proof and further a program in a typed functional language. This brief note explains this "D-Cases as proofs as programs" view and how D-Case construction and verification can benefit, under this view, from theorem proving technologies and researches on programming.

1 Introduction

The current networked society of open systems presents new problems in dependability of systems not solved by traditional techniques such as fault-tolerance. The multi-site research project JST-CREST DEOS (Dependable Operating Systems for Embedded Systems Aiming at Practical Applications) is addressing such Open System Dependability problems [10, 1].

One of the main problems is the lack of clear consensus, among a system's stakeholders, on in what sense and why a system is dependable while requirements and environments are constantly changing. As a part of DEOS, we are developing the *D-Case method* to address this [7]. A D-Case is a structured argument for dependability claims of a system, supported by evidences. The D-Case method puts this explicit argument in the focus of dependability related activities: consensus building, evidence-based assurance, securing accountability, dependability evaluation, runtime-verification, managing changes, etc. The idea is adopted and extended from the Safety Case approach in safety-critical sectors and its recent generalisation of Assurance Case notion.

The basic structure of D-Cases are taken from Goal Structuring Notation [6]. A *goal*, a claim to be substantiated, is decomposed into sub goals by an argument *strategy*, an explanation why the goal follows from the sub goals. Decomposition is repeated until a direct *evidence* can be produced for each (sub) goal. Thus a D-Case argument have a tree-like (DAG) structure of nodes.

A D-Case document will be a large document that is hard to construct and review, similarly to how a safety case is. We are developing a verification plug-in for our D-Case editor to ensure logical correctness of arguments [8].

2 "D-Cases as proofs" and D-Case verification

Our basic approach is to view arguments in D-Cases as formal proof-trees and to apply theorem proving technologies to constructions and verification of D-Cases.

Elements of a D-Case argument naturally correspond to those of a formal proof: a goal is a proposition; to decompose a goal into sub goals by a strategy is to derive conclusion from premises by an inference rule (or more generally by an application of a lemma / assumptions / axioms); an evidence directly supporting a goal is a rule admitting the goal as an axiom. Figure 1 shows the correspondence of D-

* This research is supported by JST, CREST.

Cases and proof trees in a simple example. The top goal G1 of the D-Case on the left is decomposed by strategy S1 into sub goals G2 and G3, G2 has the direct evidence E1, G3 is decomposed by S2 into G4 and G5, and G4 and G5 have direct evidences E2 and E3, respectively. In the proof tree on the right, the conclusion proposition G1 is derived by the inference rule S1 from the premises G2 and G3, G2 is an axiom by axiom rule E1, G3 is derived by S2 from G3 and G4, and G3 and G4 are axioms by rule E2 and E3, respectively. For a D-Case as a formal proof, we formalise the contents of each argument element

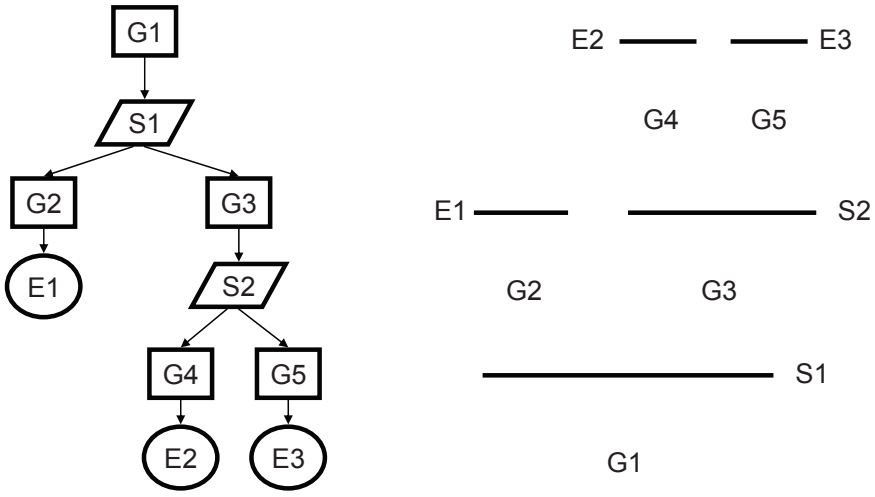

Fig. 1 Correspondence between D-Case and Proof Tree

so that machine can verify, e.g., whether a strategy is used correctly with respect to its goal and sub goals. This goes beyond the current structured-argument approach such as ARM[4], where contents of argument elements are left as unstructured text and such checks must be done by reviewing. The level of details to be formalised can be adjusted according to the benefits and costs of formalisation, ranging from conventional formal verification on rigorous models to mere format checking where necessary properties are liberally taken as axioms or assumptions.

The idea of applying theorem proving for any kind of argument is a natural one, but research targeting assurance cases/safety cases together with tool-support seems fairly recent (cf. [9, 5, 3]).

The theorem proving technology we use is Agda [2], an interactive proof assistant based on Martin-Löf Type Theory. We choose Agda to investigate the "D-Cases as programs" paradigm explained in the next sub section. Agda not only checks whether a given proof is correct, but also assist users constructing a proof by generating sub goals, by filling in routine details, etc.

Currently, a verification-plugin tool *D-Case/Agda* for the D-Case Editor is being developed. D-Case Editor and Agda system are run side by side. On the D-Case Editor side, each context / goal / strategy / evidence node of a D-Case can be annotated, besides a free-text description, by "Agda formalisation"-attribute texts. An editor command provided by D-Case/Agda tool assembles them into an Agda proof according to the tree structure of the D-Case and loads it in Agda system. User checks the module and correct errors in Agda system. D-Case/Agda tool then translates back the Agda proof to a D-Case file that D-Case Editor can graphically display and modify. Figure 2 shows such interaction between D-Case Editor and Agda.

In future versions, we plan to avoid exposing general users to Agda by

- providing generic canned patterns for D-Cases,
- parameterising the formal D-Case concept in the proof-engine, used so that a user can work on

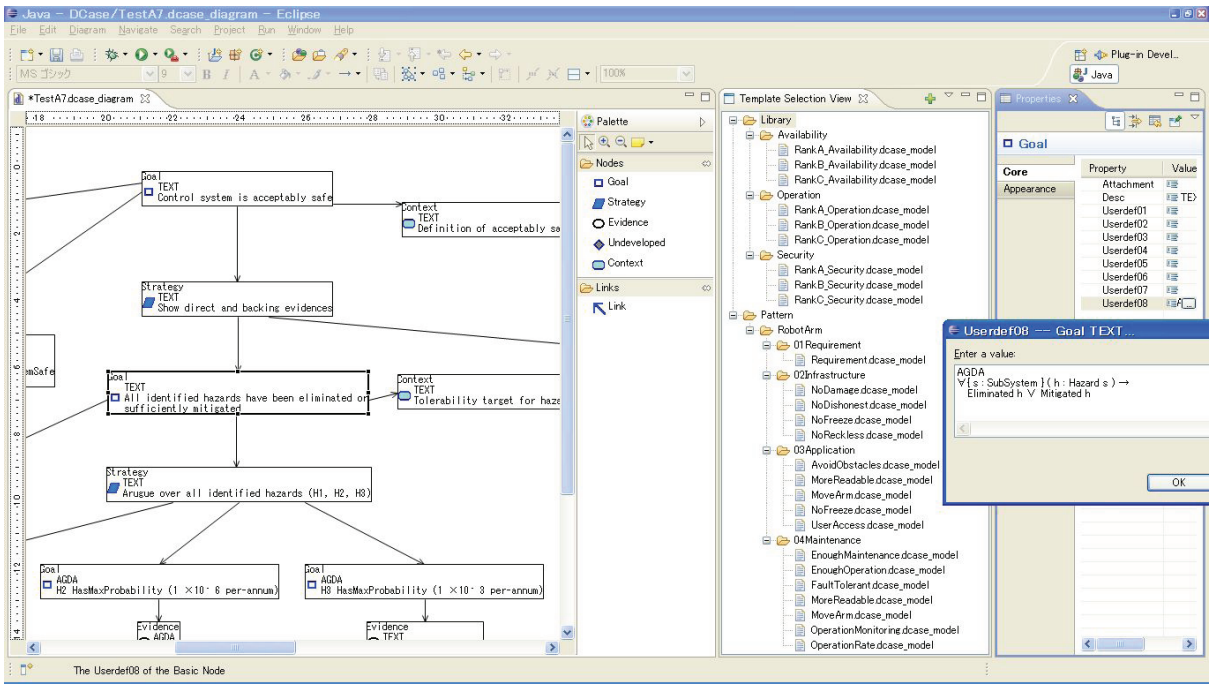

Fig. 2 D-Case/Agda

D-Case/X for her favourite engine X,

- more ambitiously, adoption of controlled natural languages with translation to/from formal counterparts,

etc.

3 "D-Cases as programs" paradigm

Theorem proving in Agda is based on the "propositions as types, proofs as programs" paradigm. This paradigm gives a correspondence between formal proofs and functional programs. With our view of D-Cases as formal proofs, this translates to "D-Cases as programs". Roughly, a goal is a type, an evidence for a goal is a piece of data of the type corresponding to the goal, and a strategy is a function that produces evidences for its goal from those of sub goals. Under the correspondence, a D-Case is logically correct if and only if the corresponding program is well-typed.

This "D-Cases as programs" paradigm opens up new possibilities of applying researches on programming and programming languages to construction and verification of D-Cases. One of our D-Case research aims is to exploit those possibilities, bringing together communities of assurance, programming, and theorem proving. Agda, on which we have been collaborating with the developers at Chalmers University of Technology, Sweden, is suited for that purpose. Unlike other spartan proof description languages, it is designed to be a general purpose programming language with modern features and gaining support from programmers.

D-Case patterns and modules are one of areas that immediately benefits from a principled programming language approach. The current informal treatment views a pattern as a D-Case with various named holes, but soon we must consider, e.g., the problems of scope, variable capture etc.

Other possible areas of investigation include:

- D-Case processes: adopting software engineering processes for D-Case development.
- Libraries and frameworks: large scale organisation for reuse, flexibility, etc. learned from software

libraries and frameworks.

- Domain specific embedded language: principled ways of deriving purpose-made dialects while keeping interoperability.
- Higher-order/Meta D-Cases: D-Cases about D-Cases assuring properties of the latter.
- Dynamic D-Case: execution of D-Cases with I/O.
- Reflective/Generative D-Cases: D-Cases inspecting parts of themselves and adopting to changes.
- Model-based D-Case development
- Program/model extraction from D-Case

References

[1] DEOS Project. http://www.crest-os.jst.go.jp/ , http://www.dependable-os.net/.

[2] The Agda Wiki. http://wiki.portal.chalmers.se/agda/pmwiki.php.

[3] Nurlida Basir, Ewen Denney, and Bernd Fischer. Constructing a safety case for automatically generated code from formal program verification information. In Michael D. Harrison and Mark-Alexander Sujan, editors, *SAFECOMP*, volume 5219 of *Lecture Notes in Computer Science*, pages 249–262. Springer, 2008.

[4] Object Management Group. Argument metamodel (ARM). OMG Document Number Sysa/10-03-15.

[5] Jon G Hall, Lucia Rapanotti, and Michael Jackson. Problem oriented software engineering, 2010. Technical Report NO2010/0329, Department of Computing, Faculty of Mathematics, Computing and Technology, The Open University.

[6] Tim Kelly and Rob Weaver. The goal structuring notation - a safety argument notation. In *Proc. of the Dependable Systems and Networks 2004, Workshop on Assurance Cases*, 2004.

[7] Yutaka Matsuno, Makoto Takeyama, Jin Nakazawa, Atsushi Itoh, Hajime Ueno, Toshinori Takai, and Hiroki Takamura. Dependability Case and metrics for Open Systems Lifecycle, version 0.4. DEOS project report, 2010.

[8] Keishi Okamoto and Makoto Takeyama. An approach to assurance cases. In *Proc. of Dependable System Workshop (DSW'09), Hakodate, Hokkaido, Japan, July 14-15*, 2009.

[9] John Rushby. Formalism in safety cases. In *Proc. 18th Safety-Critial Systems Symposium, Bristol, UK*, pages 3–17, 2010.

[10] Mario Tokoro. White paper: Dependable embedded operating system for practical use (DEOS) project, version 2. DEOS project report, 2010.

“D-Cases as proofs as programs” の考え方について (in English)
(算譜科学研究速報)
発行日：2010 年 10 月 28 日
編集・発行：独立行政法人 産業技術総合研究所 (組込みシステム技術連携研究体)
同連絡先：〒661-0974 兵庫県尼崎市若王寺 3-11-46
TEL：06-6494-8083
e-mail：informatics-inquiry@m.aist.go.jp

A note on “D-Cases as proofs as programs”
(Programming Science Technical Report)
28 Oct. 2010
(Collaborative Research Team for Verification and Specification)(CVS)
National Institute of Advanced Industrial Science and Technology (AIST)
Nakoji 3-11-46, Amagasaki, Hyogo 661-0974, Japan
TEL：81-6-6494-8083
e-mail：informatics-inquiry@m.aist.go.jp

AIST01-J00022-107