Compositional Reasoning for Processes and Data

Liam O'Reilly*
*Swansea University
Singleton Park, Swansea, SA2 8PP
csliam@swansea.ac.uk

Till Mossakowski†
†DFKI Lab Bremen
Enrique-Schmidt-Str. 5, D-28359 Bremen
Till.Mossakowski@dfki.de

Markus Roggenbach‡
‡Swansea University
Singleton Park, Swansea, SA2 8PP
csmarkus@swansea.ac.uk

Abstract

The specification language CSP-CASL allows the modelling of processes and data within a single framework. CSP-CASL allows one to use the specification structuring operators, such as parametrization and union, to create structured specifications. Here we outline proof calculi that exploit specification structure, allowing for refinement and deadlock analysis.

1 Introduction

Specification in the *large* allows to model systems using components. In this context, algebraic specification has provided a rich variety of operators that allow to build complex specifications out of simpler ones (Bergstra et al., 1990). These operators include the union of specifications, the renaming and hiding of symbols within specifications, as well as parametrized specifications. In the context of data, these structuring mechanisms with their relations to proof calculi have been studied extensively (Bidoit et al., 1998). To the best of our knowledge, we are the first to apply these ideas to state based systems. Here, we develop proof calculi for the language CSP-CASL (Roggenbach, 2006) that exploit the specification structure to ease verification. This paper summarises our work presented in (O'Reilly et al., to appear).

2 CSP-CASL

Distributed computer applications like web services, flight booking systems, and electronic payment systems involve the parallel processing of data. Consequently, these systems exhibit concurrent aspects (e.g. deadlock-freedom) as well as data aspects (e.g. functional correctness). Often, these aspects depend on each other. The algebraic specification language CASL (Mosses, 2004) alone can deal with data aspects, while the process algebra CSP (Roscoe, 2010) is quite capable of modelling concurrent systems. The mutual dependencies between processes and data, however, require an integrated language such as CSP-CASL.

A CSP-CASL specification consists of a *data part* and a *process part*. The data part establishes the data of the system, and declares the sorts, operations and predicates available. The process part then establishes the desired behaviour of the system components. Each component is specified as a CSP process, where CASL terms are used as communications, CASL sorts denote sets of communications, relational renaming is described by a binary CASL predicate, and the CSP conditional construct uses CASL formulae as conditions.

3 Structured CSP-CASL

Recently (O'Reilly et al., to appear), we extended CSP-CASL by the structuring mechanisms as known from algebraic specification. To this end, we formulated the language as a so-called institution. In order to cater for parametrization, we introduced the notion of "loose processes". As an example of structured as well as generic CSP-CASL consider the following specification of an online shopping system:

spec SHOP [*RefCl*(CUSTOMER)] [*RefCl*(WAREHOUSE)] [*RefCl*(PAYMENTSYSTEM)] [*RefCl*(COORDINATOR)] =
process *System* : *C_C*, *C_W*, *C_PS* ;
System = *Coordinator* ⟦ *C_C*, *C_W*, *C_PS* || *C_C*, *C_W*, *C_PS* ⟧
(*Customer* ⟦ *C_C* || *C_W*, *C_PS* ⟧

(*Warehouse* $[\![$ *C_W* $||$ *C_PS* $]\!]$ *PaymentSystem*))

end

The above describes how the overall *System* is composed out of four components, i.e. processes, namely *Customer*, *Warehouse*, *PaymentSystem*, and *Coordinator*. These are specified separately in specifications CUSTOMER, WAREHOUSE, PAYMENTSYSTEM, and COORDINATOR. The *RefCl* operator ensures that any refinement of these component specifications provides a valid actual parameter for SHOP. Concerning proof support, the question arises if such a parametrized specification allows for compositional reasoning.

4 Compositional reasoning

In (O'Reilly et al., to appear), we developed proof calculi that allow to decompose complex proof obligations on structured specifications into simpler ones. The rules of these calculi exploit in a systematic way the specification building operators such as union, renaming, and hiding. Astonishingly enough, even proof rules out of the CSP context are "well-behaved" under these specification building operators. Here, we study especially the notions of *responsiveness* and *deadlock-freedom of networks* (Reed et al., 2004). A typical example of one such rule in our calculi is:

$$\frac{Server :: A\ ResToLive^{\checkmark}\ Client :: B\ on\ J\ in\ SP_1}{Server :: A\ ResToLive^{\checkmark}\ Client :: B\ on\ J\ in\ (SP_1\ \mathbf{and}\ SP_2)}$$

This proof rule (for the union operator, which syntactically is written as **and**) states that in order to show that a *Server* is responsive to a *Client* in the context of the specification $(SP_1\ \mathbf{and}\ SP_2)$, it is enough to establish responsiveness in the context of SP_1 alone. Here, A, B and J denote sets of events. This rule illustrates how to reduce the specification text involved in a proof. Others rules work on the processes involved. For example, in order to prove deadlock freedom of a network, we isolate a single process, establish that it is responsive, and finally prove deadlock freedom for the smaller network with the isolated process removed. Finally, we have rules that relate refinement and structuring.

5 Proof rules in practice

An extensive specification and verification exercise on the online shop example introduced above demonstrates that our techniques work in practice. Tool support for the calculi developed is well under way.

Acknowledgements

We would like to thank Erwin R. Catesbeiana for his well structured advice on avoiding inconsistent calculi.

References

J.A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37(2):335–372, 1990.

M. Bidoit, V.V. Cengarle, and R. Hennicker. Proof systems for structured specifications and their refinements. In *Algebraic Fondations of System Specification*. Springer, 1998.

P. D. Mosses, editor. CASL *Reference Manual*. LNCS 2960. Springer, 2004.

L. O'Reilly, T. Mossakowski, and M. Roggenbach. Compositional modelling and reasoning in an institution for processes and data. LNCS. Springer, to appear.

J.N. Reed, J.E. Sinclair, and A.W. Roscoe. Responsiveness of interoperating components. *Formal Aspect of Computing*, 16(4):394–411, 2004.

M. Roggenbach. CSP-CASL: A new integration of process algebra and algebraic specification. *Theoretical Computer Science*, 354(1):42–71, 2006.

A.W. Roscoe. *Understanding Concurrent Systems*. Springer, 2010.