A Verified Translation from *Circus* to CSP_M

Artur Oliveira Gomes

Trinity College Dublin
gomesa@tcd.ie

Abstract. We are proposing the implementation of a verified tool capable of translating *Circus* programs into CSP_M allowing the user to perform model checking using existing tools such as FDR. In this paper, we discuss some existing strategies for a manual translation and then we summarise the steps required in order to produce and verify the implementation of a translation tool.

1 The Problem

Nowadays, providing software correctness is likely to be an increasing challenge for developers, as software complexity and magnitude are growing exponentially. Moreover, the software development processes recently are being required to be complete within short periods of time, which conflicts with the timing required for software verification. The problem can be, at least partially, solved with support of a validation process, such as testing, formal verification, and model checking, through techniques and tools with as much automation as possible, since such task being performed manually requires time, but more importantly, it can be as error-prone as is the software to be verified.

Circus is a formal language, which combines Z, and CSP constructs allowing both behavioural and structural aspects of systems to be captured as *Circus* specifications. At present, however, there is no tool support for model- checking *Circus*. In order to overcome this problem, we can translate *Circus* into CSP by hand, and then, model-check the translated model using FDR, which supports specifications written in CSP. FDR analyses failures and divergence models, with checks related to, for example, deadlock, livelock, and termination.

We have to translate *Circus* into CSP, by adapting the *Circus* model to the CSP view of the world. For instance, we have to capture the state-based features of *Circus*, derived from Z, in CSP. However, industrial-scale applications would require too much effort and caution from the user in order to avoid the introduction of errors due to the handmade translation, not mentioning the fact that a handmade translation is time consuming.

Recently, we have worked on formalising a model of the haemodialysis machine [12] using *Circus*. The haemodialysis machine purifies the blood of a person whose kidneys are not working normally, removing waste and excess of water from the blood. We produced a manual translation from the haemodialysis *Circus* model into CSP in order to analyse our model using FDR. Our research showed that the manual translation is not easy for large systems, and therefore an automatic tool would be useful for this task.

2 Related Works

In this section we survey a few existing formalisms and tools: we first present some state-based languages; then we detail some process-based languages; and finally we show a few combination of these formalisms. In our project we aim at a formalisms that captures both state-based constructs and concurrent processes.

Z [30], B [1] and VDM [8], are used to model structural aspects of a system, providing a mathematical description, using, for example, set theory, first-order logic and *lambda calculus*. However, these languages are not designed to model aspects of the system behaviour like communication between components.

Verification of state-based languages is possible for languages like Z, B and VDM. A Z refinement calculus is presented by Cavalcanti and Woodcock [3]. Z specifications written in LaTeX can be animated with JAZA [29]. Likewise, the Community Z Tools [5] parses Z specifications and allows a range of assessments.

The communication behaviour aspects of a system are captured by languages such as *Communicating Sequential Processes* (CSP) [15] and *Calculus of Communicating System* (CCS) [16]. With these languages, we can describe interactions, communication and synchronization between processes. Programs written using CSP be verified regarding non-determinism, *deadlock* freedom and *livelock* freedom with help of model analysis tools, via *model- checking* using FDR [10].

As we aim at the verification of complex and critical systems, we need a formalization that combines both state-transformation and communication aspects of the system, allowing us to create more complete formal models. Combinations of Object-Z along with CSP, are addressed in [26]. On the other hand, Schneider and Treharne integrate CSP and B [28]. The Z language combined with CSP is present in [19]. Finally, Galloway and Stoddart combine CCS with Z [11].

Woodcock and Cavalcanti defined *Circus* [31], which is a formalism that not only combines Z and CSP, but also Morgan's refinement calculus [17], and Dijkstra's guarded command language [6]. Its semantics is based on the *Unifying Theories of Programming* [14]. Furthermore, a refinement calculus for *Circus* is presented by Oliveira [24], using tool support with ProofPower-Z [32]. An extension of *Circus* presented by Sherif and He, is used in order to capture the temporal aspects of systems, known as *Circus Time* [25].

Currently, there are a few tools for supporting *Circus*, such as a refinement calculator, CRefine [4], and an animator for *Circus*, Joker [21]. However, model-checking tool for *Circus* is still unavailable. The usual process in order to overcome that is to translate *Circus* by hand to CSP_M, and then use FDR. Model checking through FDR allows the user to perform a wide range of analysis, such as refinement checks, deadlock and livelock freedom, and termination.

A manual translation from *Circus* into CSP_M is described by Oliveira *et al.* [23], where they apply the refinement laws to *Circus* programs in order to translate them into CSP_M. Their approach focuses on a subset of *Circus* that can be easily mapped into CSP_M, by transforming state-rich *Circus* processes into *stateless* processes. There is also work by Mota *et al.* that prototypes a strategy for model checking *Circus* using Microsoft FORMULA [18].

Another strategy used in order to overcome the lack of tool support for model checking *Circus* is proposed by Beg [2], who prototyped and investigated an automatic translation that supports a simplified abstract form of *Circus*. However, the Haskell work did not involve the development of a *Circus* parser.

Research Questions:

1) Is it possible to build a verified translator from *Circus* to CSP_M?
2) Does the use of Haskell make this task easier?
3) What precisely does it mean for the translator to be correct?
4) What is the relationship between the correctness of the translator tool in development here and the arguments in Oliveira *et. al.* [23] about the correctness of their translation scheme?

In this Ph.D project, we propose the development of a verified tool for translating *Circus* programs into CSP_M. We are developing a tool that partially automates the translation technique proposed by Oliveira *et. al.* [23]: they use the *Circus* refinement laws in order to transform state-rich *Circus* specification into a *stateless* *Circus* version of the specification. For this, the state components of a state-rich *Circus* process are transformed into a *Memory* process [20], with *get* and *set* messages capturing the state changes, resulting in a stateless *Circus* process. The translated CSP_M code will be in conformance with FDR. This tool partially automates the refinement of *Circus* programs, using the *Circus* refinement laws, as it will be required some degree of input from the user.

The entire tool set will be developed as an extension of JAZA, which parses Z specifications written in LaTeX, the same input used by the Community Z Tools. *Circus* specifications are written as a sequence of block environment, very similar to the way Z paragraphs are written. However, we assume that the *Circus* document is already type checked by existing tools [5]. Our goal is to formally verify the implementation of the translation from *Circus* to CSP_M using of Isabelle/HOL theorem prover with help of Haskabelle, a tool used to translate Haskell programs into Isabelle specifications [13]. We can take advantage of our experience with Isabelle/HOL whilst verifying our Haskell code. However, we also know that it is possible to verify Haskell programs using testing, model checking and interactive theorem proving [7].

3 Current work and next steps

As a first step towards our model-checking approach for *Circus* we produced a new parser for *Circus* specifications written in LaTeX. We expanded the JAZA parser in order to support *Circus* on top of the existing support for the Z language. We are making a step forward and bringing the *Circus* notation into the JAZA tool so it can parse *Circus* programs and then, targeting model checking.

As we have the new parser for *Circus*, we are now working on two new tasks: (1) designing the tool for automating the translation from *Circus* into CSP_M,

which should include (2) the implementation of the *Circus* refinement laws in Haskell. Task 2 looks similar to CRefine [4], however, because we are writing Haskell code, we will import that code into Isabelle/HOL and use theorem prover in order to certify that the implementation of the laws is correct.

From the tasks (1) and (2), we introduce a third one, which involves establishing precisely what we mean by a correct translation. The verification step of the implemented tool should involve: (3.1) defining the specification of the translation to be capture in Isabelle/HOL; (3.2) establishing the theorem of correctness of the implementation; and (3.3) defining the refinement relation between laws and translations from [22,23], and the corresponding Haskell code that implements those translations. We intend to show the correctness of our implementantion of the *Circus* refinement laws. By defining the refinement relation, we capture the relationship between the mathematical notation of the laws [22] and our Haskell representation of the *Circus* AST.

Oliveira *et al.* [23], suggest the use of the refinement laws as part of the translation in order to obtain CSP_M specifications from *Circus*. In their work, a sequence of refinement steps is followed, applying a some *Circus* refinement laws So far, we have implemented the selected set of refinement laws in Haskell, but the integration with JAZA is yet to be implemented.

The implementation of part of the *Circus* refinement laws in Haskell is essential in order to apply the refinement laws whilst rewriting *Circus* processes, according to the proposed translation strategy. We intend to ensure that the translation process is as automated as possible in order to produce an equivalent CSP_M specification that relies on its original *Circus* version, in order to avoid the introduction of errors in the specification due to user interaction.

The reason for adopting the translation presented by Oliveira *et al.* is that, even though it is a manual translation, with no tool support involved, each translation step is justified by the *Circus* refinement laws, and these have been formally proved to be consistent using ProofPower-Z [22]. Currently, their approach covers a subset of *Circus*. Once we have implemented that subset, we can extend it in order to cover a wider group of *Circus* constructs. In our work, we want to show that our implementation of the above mentioned translation rules and refinement laws in our translator is correct.

As the approach for model checking *Circus* consists of mapping the *Circus* into CSP. It means that, for instance, we should consider that CSP does not capture the notion of *state*. Thus, we need to provide a transformation that carries the values stored in a state-rich *Circus* processes into the CSP specifications.

Our tool transforms the specification from state-rich to stateless *Circus* processes using our Haskell representation of *Circus* using Haskell's data types, obtained from the parsing process from LaTeX. So far, we have the functions that transform state-rich into stateless *Circus* processes, defined as Ω functions, implemented but not yet integrated with JAZA. This step is similar to what will be made with the application of the *Circus* refinement laws.

Moving towards the CSP_M code, we then have to apply the mapping functions, defined as Υ functions, that translates stateless *Circus* into CSP_M. Such

task may look simple for some parts of a *Circus* specification since it includes CSP constructs. However, we have to carefully map the Z constructs into CSP, such as predicates and expressions. Up to date, we have the implementation of all the Υ functions [23]. However, our tool does not contemplate the translation of all Z expressions and predicates, and neither translates Z schemas and axiomatic definitions[1] into CSP_M, which will be addressed in our future work.

In this work, we are proposing the development of a tool that will not only be able to automatically translate *Circus* into CSP_M, but also to pretty-printing the specification into either: CSP_M, after applying the mapping functions; LaTeX, as an output for the refined specification; and, a for a future work Haskell code resulted from the refinement. Moreover, in a near future, we can integrate the tool with a *Circus* refinement calculator allowing us to refine *Circus* to *Circus*. This is similar to CRefine, however, we plan to design a tool for discharging proof obligations with support from a theorem proving [9], and also plan to refine *Circus* into a programming language, such as Haskell itself.

Acknowledgements This work is funded by CNPq (Brazilian National Council for Scientific and Technological Development) within the Science without Borders programme, Grant No. 201857/2014-6, and partially funded by Science Foundation Ireland grant 13/RC/2094.

References

1. Abrial, J.R.: The B-book - Assigning Programs to Meanings. Cambridge University Press (2005)
2. Beg, A., Butterfield, A.: Development of a prototype translator from Circus to CSPm. In: ICOSST. pp. 16–23 (Dec 2015)
3. Cavalcanti, A.L.C., Woodcock, J.C.P.: ZRC—A Refinement Calculus for Z. Formal Aspects of Computing 10(3), 267—289 (1999)
4. Conserva Filho, M., Oliveira, M.V.M.: Implementing tactics of refinement in crefine. In: SEFM'12. pp. 342–351. Springer (2012)
5. CZT: Community Z Tools (Oct 2016), `czt.sourceforge.net/`
6. Dijkstra, E.W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs. Commun. ACM 18, 453–457 (August 1975)
7. Dybjer, P., Haiyan, Q., Takeyama, M.: Verifying haskell programs by combining testing, model checking and interactive theorem proving. Information and software technology 46(15), 1011–1025 (2004)
8. Fitzgerald, J.S., Larsen, P.G.: Modelling Systems - Practical Tools and Techniques in Software Development (2. ed.). Cambridge University Press (2009)
9. Foster, S., Zeyda, F., Woodcock, J.: Isabelle/utp: A mechanised theory engineering framework. In: Naumann, D. (ed.) Unifying Theories of Programming - 5th International Symposium, UTP 2014, Singapore, May 13, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8963, pp. 21–41. Springer (2014), `http://dx.doi.org/10.1007/978-3-319-14806-9_2`

[1] We are working with predicates, expressions, schemas and axiomatic definitions currently supported by JAZA, based on Spivey's [27] Z notation.

10. FSE, F.S.E.L.: Failures-Divergence Refinement: FDR2 User Manual. (Jun 2016)
11. Galloway, A.J., Stoddart, W.J.: An Operational Semantics for ZCCS. In: ICFEM'97. pp. 293–302. IEEE Computer Society Press (1997)
12. Gomes, A.O., Butterfield, A.: Modelling the Haemodialysis Machine with Circus. In: ABZ'16. LNCS, vol. 9675, pp. 409–424. Springer (2016)
13. Haftmann, F.: From higher-order logic to haskell: there and back again. In: Gallagher, J.P., Voigtländer, J. (eds.) Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2010, Madrid, Spain, January 18-19, 2010. pp. 155–158. ACM (2010), http://doi.acm.org/10.1145/1706356.1706385
14. He, J., Hoare, C.A.R.: Unifying Theories of Programming. In: Orlowska, E., Szalas, A. (eds.) RelMiCS. pp. 97–99 (1998)
15. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)
16. Milner, R.: A Calculus of Communicating Systems. Springer (1982)
17. Morgan, C.C.: Programming From Specifications, 2nd Edition. Prentice Hall International series in computer science, Prentice Hall (1994)
18. Mota, A., Farias, A., Didier, A., Woodcock, J.: Rapid Prototyping of a Semantically Well Founded Circus Model Checker. In: SEFM'14. pp. 235–249. Springer (2014)
19. Mota, A., Sampaio, A.: Model-Checking CSP-Z. In: European Joint Conference on Theory and Practice of Software. pp. 205–220. Springer (1998)
20. Nogueira, S., Sampaio, A., Mota, A.: Test generation from state based use case models. Formal Aspects of Computing 26(3), 441–490 (2012)
21. Oliveira, D., Oliveira, M.: Joker: An Animation Framework for Formal Specications. In: SBMF'11 - Short Papers. pp. 43–48. ICMC/USP (Sep 2011)
22. Oliveira, M., Cavalcanti, A., Woodcock, J.: Unifying Theories in ProofPower-Z. Formal Aspects of Computing (2007)
23. Oliveira, M.V.M., Sampaio, A.C.A., Antonino, P.R.G., Ramos, R.T., Cavalcanti, A.L.C., Woodcock, J.C.P.: Compositional Analysis and Design of CML Models. Tech. Rep. D24.1, COMPASS Deliverable (2013)
24. Oliveira, M.V.M.: Formal Derivation of State-Rich Reactive Programs using *Circus*. Ph.D. thesis, Department of Computer Science, University of York (2005)
25. Sherif, A., Cavalcanti, A., He, J., Sampaio, A.: A process algebraic framework for specification and validationof real-time systems. Formal Asp. Comput. 22(2), 153–191 (2010)
26. Smith, G.: A Semantic Integration of Object-Z and CSP for the Specification of Concurrent Systems. In: Proceedings of FME 1997, volume 1313 of LNCS. pp. 62–81. Springer (1997)
27. Spivey, J.M.: The Z Notation: A Reference Manual. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
28. Treharne, H., Schneider, S.: Using a Process Algebra to Control B Operations. In: IFM. pp. 437–456 (1999)
29. Utting, M.: Jaza User Manual and Tutorial (June 2005), www.cs.waikato.ac.nz/~marku/jaza/userman.pdf checked May 1st, 2016
30. Woodcock, J.C.P., Davies, J.: Using Z–Specification, Refinement, and Proof. Prentice-Hall (1996)
31. Woodcock, J., Cavalcanti, A.: The Semantics of Circus. In: ZB '02. pp. 184–203. Springer, London, UK (2002)
32. Zeyda, F., Cavalcanti, A.: Mechanical Reasoning about Families of UTP Theories. In: SBMF 2008. pp. 145–160 (2008)