# Tool Support for the *Circus* Refinement Calculus

A. C. Gurgel, C. G. de Castro and M. V. M. Oliveira

Departamento de Informática e Matemática Aplicada, UFRN, Brazil

*Circus* [1] specifications combine both data and behavioural aspects of concurrent systems using a combination of CSP [3], Z [9], and Dijkstra's command language. Its associated refinement theory and calculus [5] distinguishes itself from other such combinations. Using the *Circus* refinement calculus, we can correctly construct programs in a stepwise fashion [4]. Each step is justified by the application of a refinement law, possibly with the discharge of proof obligations (hereafter called POs). Hence, using *Circus* we are able to calculate concrete (usually distributed) specifications from abstract (usually centralised) specifications. The manual application of the refinement calculus, however, is an error-prone and hard task.

We present CRefine[1], a tool that supports the use of the *Circus* refinement calculus[2]. Its interface is similar to Refine's [8], a tool that supports Morgan's refinement calculus [4]; it is based on an early prototype that was presented in [10]. We have, however, considerably changed and extended CRefine's prototype. First, we updated the *Circus* parser used which fixes a couple of bugs of its earlier version. We have also added facilities to manage developments: undoing and redoing refinement steps, saving and opening developments is now available. Furthermore, some GUI facilities like pretty-printing, filtering applicable laws according to the selected program, classification of laws, adding comments to the development, and printing the development were also included. Finally, the discharge of some proof obligations is now automatically done by CRefine.

CRefine provides support to apply the refinement laws and to manage the overall development. Its **interface** is composed by a menu and three main frames: refinement, proof obligations, and code. The refinement frame shows all the steps of the refinement process. This includes law applications and retrieving the current status of an action or process (collection). The proof obligations frame lists the POs that were generated by the law applications, indicates their current state (i.e. checked valid or invalid, or unchecked), and associates each one of them to the law application that originated it in the refinement frame. Currently, some proof obligations are automatically checked valid or invalid. In our experience, these amount to over 60% of the proof obligations. The remaining proof obligations need to be verified by the user. Finally, the code frame exhibits the overall *Circus* specification that has been calculate so far.

CRefine provides two display formats for formulas: LaTeX and Unicode (pretty-printing). We intend to use this tool in teaching the *Circus* refinement calculus to under-graduates. Unfortunately, most of them are not familiar with LaTeX; in order to make CRefine accessible to them, we have also provided a pretty-printing.

[1] Available at `http://www.cs.york.ac.uk/circus`

[2] This work is financially supported by CNPq: grant 551210/2005-2

This pretty-printing is also a success among researchers, since it unconditionally makes the presentation of the development more user-friendly.

The starting point of a **development** in CRefine is a LaTeX file that contains the abstract specification of the system to be refined. Starting from this specification, the application of refinement laws is as follows: first, we select the part of the *Circus* program that we want to refine by clicking on its lines (multiple lines can be selected by clicking on the first and last line of the term); then, we select the law we want to apply; finally, after the input of any arguments that may be required by the law, the application is automatically done. This updates the refinement frame, the proof obligations frame, and the code frame.

Law applications can be done in two ways. First, a right-click on the selected term shows a pop-up menu that lists only those laws that can be applied to the selected term. In this case, their effective application is done by selecting them in this list. Second, we can select the law from a list in the main menu that contains all the refinement laws. If the law can be applied to the term, the apply button is enabled and we can apply the law by clicking on it. When needed, an argument window is shown to the user before the application of the law. In this window, users may either type the argument in a LaTeX format, or use a symbol keyboard. The user may see the details of a refinement law by selecting it in the list of the main menu and then right-clicking on its name.

Using CRefine's **development management** users may (when applicable) undo and redo development steps. That means different development paths during a development may be tried, possibly in a search for a more efficient implementation. Developments may also be saved in order to be continued latter; CRefine's developments are saved in XML format. Users may document the development by adding, editing, and viewing comments to each term or law application in the refinement window. Finally, CRefine automatically generates a LaTeX file that documents the main elements of the development: original specification, refinement, POs, comments and the concrete specification. The user can choose which elements should be included in the final document.

CRefine's **architecture** is strongly based on the architecture proposed for *Circus* tools in [2]. It extends an ongoing effort of the Community Z Tools (CZT), which provides a set of tools for the Z specification language. In recent years, many *Circus* collaborators have made extension on the CZT project to provide tools that support *Circus* like a parser, a type-checker, a refinement model-checker, a theorem-proving module, and pretty-printers.

The cost of developments may still be reduced. Frequently used strategies of refinement are reflected in sequences of laws that are applied over and over again. Identifying these strategies, documenting them as tactics, and using them in program developments as single transformation rules brings a profit in effort. In [6], we present a refinement-tactic language called ArcAngel*C*, which can be used to formalise tactics of refinement just like in [7]. Allowing users to define and use tactics as simple refinement laws within CRefine is our next step.

The vast majority of the refinement laws from [5], which have been used in a reasonable number of case studies, are included in CRefine. This give us

confidence that the current set of laws is appropriate for useful applications. We are aware, however, that it is not complete [5]. We intend to provide a parser of refinement laws in the style of CZT. Using this parser, the laws could be dynamically loaded; no recompilation would be needed.

Another interesting piece of future work is the automatic discharge of the remaining POs, which can be predicates or action/processes transformations. For this, we need to integrate CRefine with a theorem-prover to check predicate POs and to allow multiple developments within CRefine to check POs that are action/processes transformations. For instance, users will be able to prove that $A_1$ is refined by $A_2$ by deriving $A_2$ from $A_1$ in a new development.

Finally, the infra-structure provided by tactics of refinement and multiple developments can be used to allow users to make sub-developments within larger developments. This would considerably modularise future developments.

CRefine can be a useful tool in the development of state-rich reactive systems. Our initial intention was to develop an educational tool and use it in teaching formal methods. However, during the implementation and tests, we noticed that it may as well be useful in the development of industrial-scale systems. Empirical verifications in a near future will verify this statement. For instance, we are currently developing case studies that are related to the oil industry.

## References


1. A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. A refinement strategy for *Circus*. *Formal Aspects of Computing*, **15**(2–3):146–181, 2003.
2. L. J. S. Freitas, J. C. P. Woodcock, and A. L. C. Cavalcanti. An Architecture for *Circus* Tools. In A. C. V. Melo and A. Moreira, editors, *Proceedings of the Brazilian Symposium on Formal Methods*, pages 6 – 21, 2007.
3. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
4. C. Morgan. *Programming from Specifications*. Prentice-Hall, 1994.
5. M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus*. PhD thesis, Department of Computer Science, University of York, 2005. YCST-2006/02.
6. M. V. M. Oliveira. ArcAngel*C*. Technical report, Departamento de Informática e Matemática Aplicada - Universidade Federal do Rio Grande do Norte, Natal, Brazil, February 2007.
7. M. V. M. Oliveira, A. L. C. Cavalcanti, and J. C. P. Woodcock. ArcAngel: a Tactic Language for Refinement. *Formal Aspects of Computing*, **15**(1):28–47, 2003.
8. M. V. M. Oliveira, M. Xavier, and A. L. C. Cavalcanti. Refine and Gabriel: Support for Refinement and Tactics. In Jorge R. Cuellar and Zhiming Liu, editors, *2nd IEEE International Conference on Software Engineering and Formal Methods*, pages 310–319. IEEE Computer Society Press, Sep 2004.
9. J. C. P. Woodcock and J. Davies. *Using Z—Specification, Refinement, and Proof*. Prentice-Hall, 1996.
10. M. A. Xavier, A. L. C. Cavalcanti, and A. C. A. Sampaio. Type Checking *Circus* Specifications. In A. M. Moreira and L. Ribeiro, editors, *SBMF 2006: Brazilian Symposium on Formal Methods*, pages 105 – 120, 2006.