Extending CRefine to Support Tactics of Refinement

M. S . C. Filho and M. V. M. Oliveira

Universidade Federal do Rio Grande do Norte – Brazil
madielfilho@gmail.com, marcel@dimap.ufrn.br

Abstract. *Circus* is a formal language, which is used to specify concurrent systems using concepts from Z and CSP. It has a refinement calculus, which can be used to develop software in a precise and stepwise fashion. Each step is justified by the application of a refinement law (possibly with the discharge of proof obligations). Sometimes, the same laws can be applied in the same manner in different developments or even in different parts of a single development. A strategy to optimize this calculus is to formalise this application as a refinement tactic, which can then be used as a single transformation rule. CRefine was developed to support the *Circus* refinement calculus. However, before the work presented here, it did not provide support for refinement tactics. This paper presents a new module in CRefine, which automates the process of defining and applying refinement tactics that are formalised in the tactic language ArcAngel*C*. Furthermore, we validate the extension by applying the new module in a case study, which consists in a refinement strategy for verification of SPARK Ada implementations of control systems.

Keywords: Refinement Calculus, Tactic, Tool Support, *Circus*.

1 Introduction

Circus [2] is a formal language which can be used to specify concurrent and reactive systems. It is a combination of Z [12] and CSP [11]: the former cares about the data aspects of sequential systems and the latter is specific to concurrent systems and defines the concurrent behaviour of the system. Besides the specification of data and behavioural aspects of concurrent systems, *Circus* has a refinement calculus. This calculus consists of repeated application of refinement laws to an initial abstract specification to produce a concrete specification. Using a refinement calculus, programs can be developed correctly in a stepwise fashion. Each step is an application of a refinement law, which might be valid only under certain conditions that need to be proved.

Sometimes, the same laws are applied in the same manner in various developments or even in different parts of a single development. A strategy to optimize this calculus is to formalise these applications as refinement tactics, which can then be used as single transformation rules.

The manual development using a refinement calculus is a hard and error-prone task because it encompasses many refinement laws in mostly long and repetitive developments. CRefine [8] was develop to support the application of the *Circus* refinement calculus. It automates the management of the development and its proof obligations, some of which are automatically proved. It is based on Refine [9], a tool that supports Morgan's refinement calculus [6] for sequential programs. However, the current version of CRefine does not provide support for the definition and use of refinement tactics.

This paper presents an extension to CRefine that consists of a new module that allows the definition and application of refinement tactics. The tactic language supported is ArcAngel*C* [10], a refinement tactic language for *Circus* programs that is similar to the tactic language for sequential programs ArcAngel [7]. Both tactic languages are based on the general tactic language Angel [5]. ArcAngel*C* main difference to ArcAngel is the possibility of defining tactics that can be applied to *Circus* actions, processes and programs. ArcAngel tactics can only be applied to sequential programs. ArcAngel*C* has a formal semantics, which is based on ArcAngel's semantics, but adds some generality to support all extra *Circus* structural combinators.

The next section presents the new module of CRefine that supports use of refinement tactics. Section 3 presents the use of CRefine and its extension in a case study. Finally, in Section 4 we make our final considerations and discuss future work.

2 CRefine extension

CRefine was developed to support the *Circus* refinement calculus by automating the management of the overall development. Its main function is to apply refinement laws, sometimes with the discharge of proof obligations (hereafter called POs). Some of the POs are discharged automatically.

CRefine GUI has a menu and three main frames [8]. The first frame displays all the refinement steps of the development. The current result of the refinement process is displayed in a second frame. Finally, the last frame lists all POs that have been generated in the development and marks them as valid (automatic proof), invalid, or unknown. POs marked as unknown need to be verified by the user.

Using CRefine the user starts with the LaTeX document that contains the abstract specification and repeatedly applies refinement laws. The law application consists of selecting the term in the development frame, and choosing an applicable law in the pop-up menu list. Some law applications requires arguments that are also given by the user. Afterwards, the law is automatically applied: CRefine updates all frames based on the result of the application.

For optimisation purposes, we developed a module in CRefine that makes it possible to create tactics and use them in a program development. This was achieved by adding some new functionalities that we describe in the sequel.

2.1 Using Tactics

CRefine's GUI was modified to allow the use of refinement tactics. The changes include the addition of a new item to the menu, *ArcAngelC*, which gives access to a tactic editor in which users can create, edit and delete tactics.

The user creates a tactic using the tactics editor. The tactics are written in LaTeX and compiled within the editor (specific LaTeX commands are used for ArcAngel's constructs). The successful compilation results in the addition of the tactic to CRefine; the tactic may then be used as a single transformation rule. Using the editor, tactics may also be edited. Finally, we can also remove a previously created tactic.

Tactic application can be achieved in the same way as for law application: term selection followed by the tactic selection, possibly with the input of tactic arguments. If the tactic application succeeds, all frames are updated.

2.2 CRefine's Architecture

The extension to the original architecture of CRefine [8] did not require integration with any further external frameworks. However, new components were added to the tool's architecture.

In our extension, we added the module `Tactics`, which is responsible for creating and applying refinement tactics. In this new module, we added an ArcAngel*C* parser that directly follows the ArcAngel*C* syntax. The package `Apply` provides support for tactics application. Each ArcAngel*C* construct was transformed into a Java class that inherits (possibly indirectly) from `TacticComponent`, an abstract class with a single method, `apply`. Most of the classes implement this method following directly the ArcAngel*C* semantics [10]. The only exception is the recursive tactic as we explain in the sequel.

For pragmatic reasons, we introduce a tactical for recursive tactics that imposes an upper limit of unfoldings that are performed. The tactical μ_A monitors the number of iterations performed, behaves like **abort** if a certain threshold n is reached. If we do not want to treat non-termination as an abnormal case, but use it to control the behaviour of other tactics, the alternative tactical μ_F yields failure rather than abortion when exceeding the threshold.

The user might create his own tactics. For that, he writes the tactic using the specific editor and compiles it. Internally, this compilation involves the parsing and further verification like the existence of the refinement laws used and the validity of the tactic declaration and arguments used. Finally, if the compilation is successful, the system stores the tactic and sends a successful message to the GUI. Otherwise, an error message is shown.

The tactic application starts with the user selecting a term. The Tactic module provides CRefine with a list of tactics that is displayed to the user along with the laws that are applicable to the term selected. If a tactic is chosen, the method `apply` of the tactic is invoked using the selected term as argument. Each implementation of the method `apply`, verifies if the selected term fits the structure expected by the tactic. For instance, the Java class that corresponds

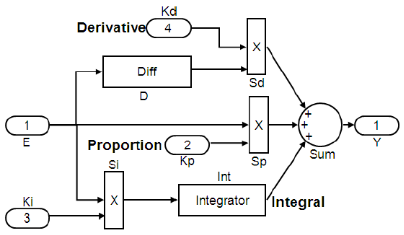

Fig. 1. PID (Proportional Integral Derivative) controller and Differentiator

to the tactic $t_1 \boxed{;} t_2$ verifies if the selected term is a sequential composition of either *Circus* actions or processes. If this is the case, the tactics t_1 and t_2 are applied to each part of the sequential composition. The result is used by CRefine to update all CRefine's frames accordingly. If the selected term is not a sequential composition, the tactic application fails; this is informed to the user.

The extension presented here required the implementation of 78 classes in the presentation and data layers. The management layer was also extended with further 84 new methods that provide support for tactic application following CRefine's initial architecture.

3 Case Study

Control systems are often used in safety-critical applications and their verification has been of great interest. In [1] is presented an approach in which they aim at proof of correctness of code, as opposed to validation of requirements or designs. They give a semantics to discrete-time Simulink diagrams using *Circus*, and propose a verification technique for parallel Ada implementations.

A simple example of a Simulink diagram is presented in Figure 1; it contains a PID (Proportional Integral Derivative) controller, a generic control loop feedback mechanism that attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly.

Control systems present a cyclic behaviour. In [1] consider discrete-time models, in which inputs and outputs are sampled at fixed intervals. The inputs and outputs are represented by rounded boxes containing numbers. In this case study, there are four inputs, E, Kp, Ki, and Kd, and one output, Y.

Typically, a block takes input signals and produces outputs according to its characteristic function. For instance, the circle is a sum block, and boxes with a $\times$ symbol model a product. There are libraries of blocks in Simulink, and they can also be user-defined. Boxes enclosing names are subsystems; they denote control systems defined in subordinated diagrams of the model.

In the *Circus* model of the diagram, each block is represented by a process, and the diagram by a parallel composition of such processes. A more detailed account of this model and the full example are given in [10]. We formalize the first two phases: NB and BJ in [10]. In this formalisation, we split each phase of the strategy into small steps. For instance, the phase NB is split into steps NBStep1 to NBStep8. Using CRefine, we mechanised the first phase, NB, and applied it to all blocks of the case study. The NB phase refines each of the blocks by rewriting their main action to a normal form. It removes the parallelism between the actions that model the flows of execution, *Flows*, and the state update, *StUpdt*, and promotes the local variables of the main action to state components.

The tactic NB includes 31 tactics. The overall application of the tactic has 184 refinement law applications. The result was accomplished in 7 seconds in a Core 2 Duo machine with 4GB of RAM. The tactic application generated 76 POs; 70 were automatically discharged.

4 Conclusion

In this paper, we present an extension to CRefine, a tactic module, that allows the definition and use of refinement tactics in a program development as a single transformation rule. CRefine tactics are defined using a refinement-tactic language for *Circus* programs, ArcAngel*C*.

The extension of CRefine has been validated using a industrial case study, which consists in the application of a refinement strategy to verify SPARK Ada programs with respect to Simulink diagrams using *Circus*. We mechanised the first phase of this strategy, NB and applied to all components of the PID controller. This case study involved 17 ArcAngel*C* constructs including basic tactics, tacticals, various structural combinators, and program tactics. The remaining 22 constructs have been tested with unit tests.

Our case study was proposed by QinetiQ, and its implementation is representative of the architectural pattern used for the development of their safety-critical applications in avionics. The verification process adopted by QinetiQ already uses Z and CSP independently to check different aspects of these systems, namely, functionality and scheduling, separately. *Circus* and the refinement strategy that we formalise allows the verification of those aspects as part of a single formal argument. The refinement technique was developed in conjunction with QinetiQ. With the use of *Circus*, we have managed to enlarge the set of properties and systems that can be checked, without increasing the proof burden, and therefore, the costs. QinetiQ intends to use our strategy (and tools) in the verification of some of their safety-critical systems.

Future Work. We are currently working on the mechanisation of the tactic that corresponds to the phase BJ of the refinement strategy discussed here. This tactic will involve over 45 tactics and dozens of law applications. The current tactic editor accepts only ascii characters; users must type the corresponding LaTeX commands to create and edit their tactics. In a near future, we intend to provide a Unicode editor in which users may use the non-ascii symbols of ArcAngel*C*, like for instance, the structural combinators.

CRefine intends to be part of a development framework for *Circus* users. That means that using CRefine, users will be able to develop executable code from an abstract *Circus* specification. For that, an integration of all *Circus* tool initiatives is needed. Besides CRefine, the *Circus* model-checker and theorem prover [4], the *Circus* type-checker [13], an improved version of the *Circus* code generator presented in [3], and a *Circus* animator currently under development will be part of the *Circus* framework. This integration will provide *Circus* with a complete IDE that will foster the use of *Circus* for software and hardware development.

Acknowledgments

Alessandro Gurgel has originally worked on CRefine and helped to solve some issues related to the tool's extension. Leo Freitas has provided some insights related to the CZT. INES and CNPq partially supports the work of the Marcel Oliveira: grants 573964/2008-4, 476836/2009-3, 560014/2010-4. The work of Madiel is supported by CAPES.

References

1. A. L. C. Cavalcanti, P. Clayton, and C. O'Halloran. From Control Law Diagrams to Ada via *Circus*. *Formal Aspects of Computing*, 2011. Online first.
2. 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.
3. A. F. Freitas and A. L. C. Cavalcanti. Automatic Translation from *Circus* to Java. In *FM 2006: Formal Methods*, pages 115 – 130, 2006.
4. L. Freitas, A. L. C. Cavalcanti, and J. C. P. Woodcock. Taking our own medicine: Applying the refinement calculus to state-rich refinement model checking. In *8th International Conference on Formal Engineering Methods, ICFEM 2006*.
5. A. P. Martin, P. H. B. Gardiner, and J. C. P. Woodcock. A Tactical Calculus. *Formal Aspects of Computing*, **8**(4):479–489, 1996.
6. C. Morgan. *Programming from Specifications*. Prentice-Hall, 1994.
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, A. C. Gurgel, and C. G. de Castro. CRefine: Support for the *Circus* Refinement Calculus. In *6th IEEE International Conferences on Software Engineering and Formal Methods*, 2008.
9. 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.
10. M. V. V. Oliveira, F. Zeyda, and A. L. C. Cavalcanti. A tactic language for refinement of state-rich concurrent specifications. *Science of Computer Programming*, 76(9):792 – 833, 2011.
11. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall Series in Computer Science. Prentice-Hall, 1998.
12. J. C. P. Woodcock and J. Davies. *Using Z—Specification, Refinement, and Proof*. Prentice-Hall, 1996.
13. 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.