Overview of the ParTypes Artifact

Hugo A. López, Eduardo R. B. Marques, Francisco Martins,
Nicholas Ng, César Santos,
Vasco Thudichum Vasconcelos, Nobuko Yoshida

June 9, 2015

Get the latest version of this document from http://download.gloss.di.fc.ul.pt/ParTypes/ArtifactOverview.pdf. For an online tutorial that shows how to test protocol formation and generate protocols in VCC format see http://gloss.di.fc.ul.pt/tryit/tools/ParTypes.

1 Getting Started Guide

We provide an artifact that contains our software ready to use. The artifact includes an Eclipse Luna distribution, the Z3 SMT solver, and an Eclipse workspace with examples. It runs on top of Windows 7 and can be downloaded from http://download.gloss.di.fc.ul.pt/ParTypes/ParTypes.zip.

Installing the artifact

1. Install VCC software. The instructions on how to install VCC can be found at https://vcc.codeplex.com/wikipage?title=Install;
2. Download the ParTypes artifact from http://download.gloss.di.fc.ul.pt/ParTypes/ParTypes.zip. Its MD5 checksum is fdad9662fdb1dc4a95b1165df6ac8623;
3. Unzip the file to C:\;
4. Open Windows 7 menu and write path in the Search programs and files text box;
5. Select Edit environment variables for your account from the results;
6. Find the Path variable from the System variables list and select it;
7. Press the edit button and append text ;c:\ParTypes\software\ParTypesVCC\bin; c:\ParTypes\software\z3-4.3.2-x86-win\bin to the Variable value text box field. Press the OK to accept the changes.

Testing the artifact

1. Start the Eclipse IDE, whose icon is found in the C:\ParTypes folder;
2. In the Eclipse workspace you find several projects. Let us concentrate on the fdiff project, the project that contains the running example in our paper. Open file fdiff.prot, the protocol of the finite differences example as in Figure 2 of the paper (a copy of which is conveniently located at the desktop of ParTypes VM);
3. Select the file fdiff.prot in Package Explorer view and press the Compiles the protocol to VCC format icon from the tool bar (the blue "play" icon). Our plug-in verifies that the protocol is well-formed (making use of Z3) and generates file fdiff.h in the src-gen folder. The ParTypes plug-in will validate it and generate the corresponding protocol in VCC notation (as we described in Appendix A of the paper);
4. Now open file fdiff-annotated.c. This file contains an annotated version of the finite differences program (as shown in Figure 1 of the paper);
5. To run our verifier select the fdiff-annotated.c file from the Package Explorer view and press the Run VCC Verification icon (the second play icon from the tool bar). Alternatively, select menu Run $\rightarrow$ External Tools $\rightarrow$ VCC Verification from Eclipse menu bar; notice the console messages that attest that our annotated program is verified correctly (*Verification of main succeeded.*).

The workspace contains other examples that we took from textbooks, annotated, and verified, as described in Section 5 of the paper. You may open the various annotated files (e.g., pi-annotated.c) and confirm that they all verify successfully. For each example, we also include a variant of the annotated program that contains an error (marked with ERROR in the code, e.g., see fdiff-with-error-annotated.c); VCC will output failed assertion messages for these variants.

In complement, you may browse in Eclipse the source code of the ParTypes Eclipse plugin (the di.gloss.partypes.* projects), and also access the source specification of the ParTypes VCC library by following the ParTypes VCC library shortcut in the Windows Desktop.

The next section gives a detailed description of how to use our tools.

2 Step-by-Step Instructions

In order to check a program against a protocol, we need a program, a protocol, and a program verifier. Programs are written in the C programming language and make use of the MPI library interface [2]; protocols are written in a language described in the online tutorial (http://gloss.di.fc.ul.pt/tryit/tools/ParTypes); the verifier is VCC [1]. The running example for this section calculates an approximation of the value of π via numerical integration. π is the value of the area under the graph of a certain function in the interval $[0..1]$. The algorithm works as follows:

- Divide the interval $[0..1]$ in a number of subintervals;

- Let processes know the number of subintervals with a broadcast operation (MPI_Bcast);
- Each process calculates a partial sum;
- Add all the partial sums together to get an approximation for the value of π with a reduction operation (MPI_Reduce).

Process rank 0 decides on the number of intervals and broadcasts this value among all processes. Each process calculates its local sum. In the end, a reduce operation adds all the partial sums and delivers the result at process rank 0. This process may then print the result.

1. In Eclipse, select the pi project from the Project Explorer view and expand it. The project contains the files pi.prot, pi-annotated.c, pi-with-error-annotated.c, and pi.c. The first file includes the protocol of the pi program; file pi-annotated.c contains the annotated version of pi.c used for verification; file pi-with-error-annotated.c contains an annotated version of pi.c that will fail to verify; and, finally, file pi.c comprises the original pi program file, adapted from [3];
2. Open file pi.prot. Protocols are introduced with the keyword protocol followed by the protocol name. In this case there are two operations in sequence. The first, broadcast 0 integer, says that process rank 0 broadcasts an integer; the second, reduce 0 sum float, collects a floating point number from each process, sums them up, and delivers the result to process rank 0.
3. Select the pi.prot file in the Package Explorer and press the Compiles the protocol to VCC format icon from the tool bar (the blue "play" icon). In this manner, the ParTypes plug-in generates file fdiff.h in the src-gen folder, containing the protocol file written in the VCC format;
4. Open file pi-annotated.c. This file contains the annotated version of the pi program, ready to be verified. Compare pi-annotated.c and pi.c to check the required changes, and confirm that the annotations are as we describe in Section 4 of the paper, i.e., VCC limitations forces us to adapt the C source code. In particular, VCC does not support floating point arithmetic and functions with variable number of arguments (e.g., printf). Thus, we must filter out in pi.c the lines that contain printf, scanf, and also all those that contain floating point operations (assignments to variables h, sum, x, and mympi). This type of code can be compiled normally, but otherwise be ignored by VCC if we enclose it within #ifndef _PARTYPES ... #endif sections. In this process, we should preserve the control structure of the program, plus every MPI call and variable declaration. The resulting program must still compile and exchange the messages the original program was intended to.
5. To check the conformance of the C code against the protocol select the pi-annotated.c program in the Package Explorer view and press Run VCC Verification (the green "play" icon in the tool bar). VCC reports that the verification process executed successfully.

6. Now change the program. Choose 1 rather than 0 as the root process for the MPI_Bcast operation in pi-annotated.c; this corresponds to the error introduced in the pi-with-error-annotated.c example file. You may now observe that VCC verification (Run VCC Verification as explained int the previous step) outputs failed assertion messages. For the particular error at stake, you should get the following message:

   Assertion '(1) == intBcastRoot(`head)' did not verify.

7. Alternatively, you may change the root process of the broadcast in the protocol specification, for instance use 1 instead of 0 as the process root for the broadcast operation in pi.prot. After regenerating the protocol in VCC format (Compiles the protocol to VCC format button), the verification of pi-annotated.c (Run VCC Verification button) will fail with the following message:

   Assertion '(0) == intBcastRoot(`head)' did not verify.

References

[1] COHEN, E., DAHLWEID, M., HILLEBRAND, M., LEINENBACH, D., MOSKAL, M., SANTEN, T., SCHULTE, W., AND TOBIES, S. VCC: A practical system for verifying concurrent C. In *TPHOLs*, vol. 5674 of *LNCS*. Springer, 2009, pp. 23–42.

[2] FORUM, M. *MPI: A Message-Passing Interface Standard—Version 3.0*. High-Performance Computing Center Stuttgart, 2012.

[3] GROPP, W., LUSK, E., AND SKJELLUM, A. *Using MPI: portable parallel programming with the message passing interface*. MIT press, 1999.