Model Checking C# Code: A Translation Approach

Huiquan Zhu
School of Computing
National University of Singapore
huiquanz@comp.nus.edu.sg

***Abstract*—Extracting model from source code helps to ensure the implementation in accord with design. The properties of interest can be checked on implemented system via the extracted model. Previous approaches usually abstract the source at the level of intermediate language or assembly code. We are building a module to automatically extract CSP# model from C# source code and use PAT (Process Analysis Toolkit) to check the properties. As PAT support user-defined C# data type, We make the extracted model adapting to either program-implied or user-defined abstraction level.**

***Keywords*-Model Checking; Source Code; Refinement;**

I. Introduction

Complex control systems often confront subtle errors related to multiple processes cooperation. In many cases, these defects are hard to be reproduced or located. Good design helps to ensure the system satisfies the security requirements. However, defects might be introduced in the coding phase. To ensure the system fulfils the requirement we need more automatic techniques to compare the design model and the implemented system.

Directly model checking on source code is a competitive way to ease these problems. Different from testing, model checking approach has full control on the processes schedule. The trace to the error, including system and environment information, can help analysing the cause of error. It should also be configurable to thoroughly traverse all possible running paths at appropriate detail levels.

Model checking source code faces challenges. (1) The state of program contains the global or local variable, one need efficient and flexible way to represent them to avoid explosion. (2) Modelling each line of assembly code as an event in model may be inefficient, we need good abstraction from the source code, with optimization for the program language. (3) The model shall be able to check the requirement, as properties, on the design model and on the source-extracted model.

II. Related Works

Directly checking C source code is particularly interested in embedded system. There are quite some C source code model checkers today, such as BLAST, SLAM and CBMC (refer to [1], [2]). Some of them translate the source code to boolean program or CIL(C Intermediate Language), along with theorem provers and SAT solvers to check abstracted predicates. Some of them translate C source code to be the input of general model checkers such as SPIN.

Compared to C language, Java, C# and other similar object-oriental languages provide built-in multi-thread feature and garbage collect mechanism. Java PathFinder [3] and MoonWalker [4] provide two virtual machine approaches to model check Java and C# program respectively. Java PathFinder acts as a Java virtual machine so the compiled Java program can run on it, while Java PathFinder analysing the program and interleaving the operations of different threads. MoonWalker works similar to Java PathFinder but targets .NET framework. Most of the discussed systems above work on the byte-code or intermediate language level. In [5] each line of Java source code is translated to one statement of promela.

PAT (Process Analysis Toolkit) is a flexible general-purpose model checker [6], [7]. It bases on the classic process algebra CSP and extends to support share variables, asynchronous message passing, automated refinement checking [8]. The principle ideas share commons with TCOZ [9], [10] and other integrated specification languages. PAT adds real-time system module, probability model and web-service design module to allow specific domain modelling. In general, PAT accepts the models that can be represented as labelled transition system (LTS). The edge connecting the states in LTS could be abstract event or data operation. Verification of linearizability, time refinement checking and parallel verification are well supported by PAT [11], [7].

Fairness of different levels are supported by PAT, including strong/weak event level, strong/weak process level and global level [7]. PAT also supports fairness on the process counter abstraction, which enables more efficient checking on large scale parameterized systems.

PAT is developed in C# under .NET framework. The input model can import data types from standard or user-specified C# class libraries. The C# code can be attached to the events as "*event*{*code1; code2;*}". Compared to other general-purpose model checkers, PAT gives more flexibility on the model abstraction level and it shows good performance compared to state-of-art general model checkers [12].

III. EXTRACT CSP# MODEL FROM C# SOURCE CODE

Currently we are building a module of PAT to automatically extract CSP# model from C# source code. This will avoid manually translating program to the input of general-purpose model checkers, which is error-prone and time-consuming. The developers can also use it to check whether the implemented system refines the higher level design. The following features are introduced to gain the module's better performance.

Deciding the atomicity of the modelling is important as it influences the performance significantly. If we translate everything to byte-code level, it will be very specific but not quite necessary. Partial order reduction and other reduction techniques could be used after extracting the byte-code level model, but abstract the model at earlier phase will be more profitable. The consecutive operations, if they do not access outside the thread's local data, are grouped in one code block attached to one τ event with a block of C# code operating on the share variables. If the code access other objects' methods (also in the global share variables), whether the method call is modelled as an event or a process is based on the object's abstraction level(could be user-specified). For example the code "*loc_assign1; loc_assign2; methodcall1; loc_assign3;*" might be translated to "$\tau\{$*loc_assign1; loc_assign2;*$\}$ $\rightarrow$ *methodcall1();* $\tau\{$*loc_assign3;*$\}\rightarrow$ *Skip;*" or "$\tau\{$*loc_assign1; loc_assign2; methodcall1; loc_assign3;*$\}$ $\rightarrow$ *Skip;*" The code in the curly bracket will be treated as atomic.

The polymorphism is hidden in the C# libraries imported in model. This avoids manually maintaining the complex inheritance relations and thus simplifies the model. More specific, the embedded C# libraries (as .dll files) contain the definitions of the classes and their relations. For the methods of certain class, if the atomicity is defined at method level, the translated code will be put in the C# library. Otherwise the method need to be explicitly modelled by CSP# events for interleaving.

The translation approach in [5] uses variables to monitor the locks and the returned values of the methods. As for PAT we prefer using channels and events to achieve the same result. This avoids the variables corresponding to synchronization to be represented in global state. This will also save the storage of system state and enhance the extracted model's processing efficiency.

PAT supports verification of various properties, including deadlock, reachability, LTL, refinement, divergence-free etc. The assertions in the source code will be translated to an "assert" event in the model. More complex properties can be separately modelled as predicate assertions in PAT model.

Compared to other discussed approaches, our solution makes use of the PAT's embedded C# library to provide more efficient handling on the variables in the program. In the translation we allow more control on atomicity of the abstracted model. Currently the system configuration need to be a close-system. In the future we plan to add in more static optimizations or integrate it with testing framework.

REFERENCES

[1] B. Schlich and S. Kowalewski, "Model checking c source code for embedded systems," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 11, no. 3, pp. 187 – 202, 2009.

[2] D. Beyer, T. A. Henzinger, and G. Thoduloz, "Configurable software verification: Concretizing the convergence of model checking and program analysis," in *In Conf. on Computer Aided Verification (CAV)*.

[3] F. Lerda and W. Visser, "Addressing dynamic issues of program model checking," in *Proceedings of the 8th international SPIN workshop on Model checking of software*. Toronto, Ontario, Canada: Springer-Verlag New York, Inc., 2001, pp. 80–102.

[4] N. H. A. D. Brugh, V. Y. Nguyen, and T. C. Ruys, "MoonWalker: verification of .NET programs," in *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, ETAPS 2009*.

[5] K. Havelund and T. Pressburger, "Model checking java programs using java pathfinder," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 2, no. 4, pp. 366 – 381, 2000.

[6] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: towards flexible verification under fairness," in *Proceedings of the 21th International Conference on Computer Aided Verification (CAV'09)*.

[7] J. Sun, Y. Liu, J. S. Dong, and H. H. Wang, "Specifying and verifying event-based fairness enhanced systems," in *Proceedings of the 10th International Conference on Formal Engineering Methods (ICFEM 2008)*.

[8] J. Sun, Y. Liu, and J. S. Dong, "Model checking CSP revisited: Introducing a process analysis toolkit," in *Proceedings of the Third International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2008)*.

[9] B. Mahony and J. S. Dong, "Timed communicating object Z," *IEEE Transactions on Software Engineering*, vol. 26, no. 2, pp. 150–177, 2000.

[10] ——, "Blending Object-Z and Timed CSP: an introduction to TCOZ," in *Proceedings of the 20th international conference on Software engineering(ICSE'98)*. IEEE Computer Society, 1998, pp. 95–104.

[11] Y. Liu, J. Sun, and J. S. Dong, "Scalable multi-core model checking fairness enhanced systems," in *Proceedings of the 11th IEEEInternational Conference on Formal Engineering Methods (ICFEM 2009)*.

[12] J. Sun, Y. Liu, A. Roychoudhury, S. Liu, and J. S. Dong, "Fair model checking with process counter abstraction," in *Proceedings of the Second World Congress on Formal Methods (FM'09)*.