Algebraic Semantics for C++11 Memory Model

Lili Xiao[1], Huibiao Zhu[2], Mengda He[3], and Shengchao Qin[3]

[1] Donghua University, Shanghai, China
[2] East China Normal University, Shanghai, China
[3] Teesside University, Middlesbrough, UK

Abstract

The C++11 standard introduced a language level weak memory model (i.e., the C++11 memory model) to improve the performance of the execution of C/C++ programs, and the linearizability is an important issue for this model. Algebra is well-suited for direct use by engineers in symbolic calculation of parameters. It is a challenge to investigate the algebraic semantics for C++11, aiming to support the linearizability of it.

Inspired by the promising semantics, in this paper, we explore the algebraic laws for the C++11 memory model, including a set of sequential and parallel expansion laws, which are implemented in the rewriting engine Maude. We introduce the concept of guarded choice, and every program can be converted into the head normal form of guarded choice. Then the linearizability of the C++11 memory model is supported. In addition, we define a read mechanism for each variable in the generated configuration sequences, which has the ability to handle the relaxed and release/acquire accesses. Consequently, the valid execution results of any program under this memory model can be provided, based on the achieved algebraic laws.

1 Introduction

Modern multi-processors and programming languages employ relaxed (*aka weak*) memory models for efficiency reasons. The C++11 memory model is always defined as an axiomatic memory model, which does not execute stepwise. It formalizes the valid results of any program as execution graphs, which need to conform to a set of coherence axioms. The promising semantics (PS) by Kang et al. [3] provides the SC-style operational semantics for C++11, on the basis of the concept of promise and time stamp. A thread T may promise to write a value v to a memory location x at some point in the future. It enables other threads to read from it before the write is actually executed. However, for preventing out-of-thin-air behaviors, the promise must be fulfilled later. For each thread T, it owns a map from any location x to the largest time stamp of a write to x that T has observed or performed. This map can be regarded as T's view of memory. When T reads from x, it can only read from the message whose time stamp recorded for x is larger than or equal to that in T's view. If T wants to write to x, it must pick a time stamp that is strictly larger than that recorded for x in its view.

Unifying Theories of Programming (*UTP*) was developed by Hoare and He in 1998 [2]. It targets at proposing a convincing unified framework to combine and link operational semantics, denotational semantics and algebraic semantics. Each of the semantics has distinctive advantages for theories of programming. For instance, the algebraic semantics is well suited in symbolic calculation of parameters and structures of an optimal design. In this paper, aiming to support the linearizability of the weak memory model, we explore the algebraic laws for the C++11 concurrency model, including a set of sequential and parallel expansion laws. Our investigation is inspired by the promising semantics. The concept of guarded choice and head normal form is introduced, and every program can be expressed in the form of guarded choice. Then the linearizability of this memory model is supported. In addition, we implement the algebraic laws in the rewriting engine Maude. Finally, we give the definition of the read mechanism for the variables read in the generated sequences. The read mechanism can get all the valid executions, and it does not rely on the introduction to time stamp and a variety of relations appearing in the traditional execution graphs of C++11. The framework of this work is illustrated in Figure 1.

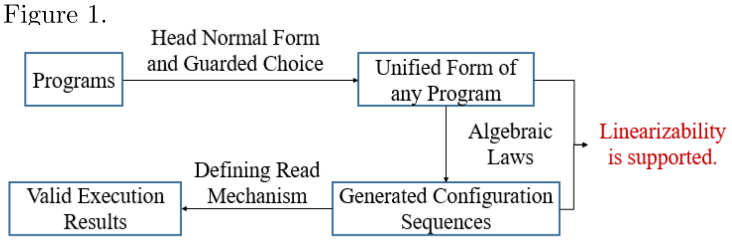

Figure 1: The framework of our work.

This paper extends our previous work presented in COMPSAC 2022 [4], where we only proposed the algebraic laws of the C++11 memory model. Now, a read mechanism is defined to generate all the valid execution results of a program under C++11.

2 Algebraic Semantics

This section aims to explore a set of algebraic laws of the C++11 memory model, including sequential and parallel expansion laws. In our approach, every program can be expressed as the head normal form of the guarded choice. Then the linearizability of C++11 is supported.

Guarded Choice. Now, we introduce the concept of guarded choice. A guarded choice is composed of a set of guarded components. $h\&(act, tid, idx)[q] \looparrowright P$ is a guarded component, where, h is a Boolean condition, and act indicates the action extracted from one statement, and we use tid to stand for the identity of the thread which performs act, and idx denotes the location of act, and the operator $\looparrowright$ is used to connect the configurations with original indices.

Head Normal Form. Next, we assign every program P a normal form called *head normal form*, $HF(P)$, which is in the form of guarded choice. Our later introduction to the read mechanism is based on the head normal form. For a relaxed write to the memory location x, it can be simulated by first promising to write to x, and then fulfilling the mentioned promise, which is left in the square brackets. And the order between the two operations cannot be broken: $HF(x := e) =_{df} [\!]\{\text{true}\&\underline{(\langle x = e\rangle, \lambda, \langle 1\rangle)}[\underline{(x = e, \lambda, \langle 2\rangle)}] \looparrowright E\}$.

Algebraic Laws. In this part, we study the algebraic laws for the C++11 memory model. Based on these laws, every program can be converted to a guarded choice.

Law (seq-1) is responsible for transferring a program into a variety of configurations statement by statement. The configurations achieved above do not depend on each other. Law (seq-2) makes them form configuration sequences, whose indices can reflect the program order. Law (seq-3) is used to generate all the valid configuration sequences of a program.

$$\begin{aligned}
\textbf{(seq–3)} \quad & (c_{11} \to c_{12} \to ...c_{1n_1}) \hookrightarrow ...(c_{m1} \to c_{m2} \to ...c_{mn_m}) \\
= & c_{11} \to \boxed{(c_{12} \to ...c_{1n_1})} \hookrightarrow ...(c_{m1} \to c_{m2} \to ...c_{mn_m}) \\
& [\!]\ ... && \text{if } d_i \\
& [\!]\ c_{m1} \to (c_{11} \to c_{12} \to ...c_{1n_1}) \hookrightarrow ... \boxed{(c_{m2} \to ...c_{mn_m})} && \text{if } d_m
\end{aligned}$$

Compared with $\looparrowright$, $\hookrightarrow$ links the configurations whose indices can reflect the program order relation. Further, $\to$ has the ability to make the configurations connected by it not be reordered.

The condition d_i indicates that the action in c_{i1} is a promise and all the configurations in front of c_{i1} are not of the release fence instructions and release writes. It is formalized as below.

$$d_i =_{df} \forall c \bullet \left(\begin{array}{l} \pi_1(c_{i1}) \text{ is in the form of } \langle x = e\rangle \wedge \\ \left(\begin{array}{l} \pi_3(c) < \pi_3(c_{i1}) \to (\pi_1(c) \text{ is not in the form of fence-rel} \wedge \\ \pi_1(c) \text{ is not in the form of } x_{\text{rel}} = e) \end{array} \right) \end{array} \right)$$

In addition, the parallel expansion law (par-1) regards the parallel model as a variant of the interleaving model.

3 Read Mechanism

In this section, we introduce the read mechanism, which can be applied in the configuration sequences of C++11 generated from our algebraic laws. Then all the valid executions of a program can be produced. The framework of the read mechanism is exhibited in Figure 2. Also, the method used here is able to throw away the concept of time stamp and various relations such as hb and rf relations in the traditional execution graphs of C++11.

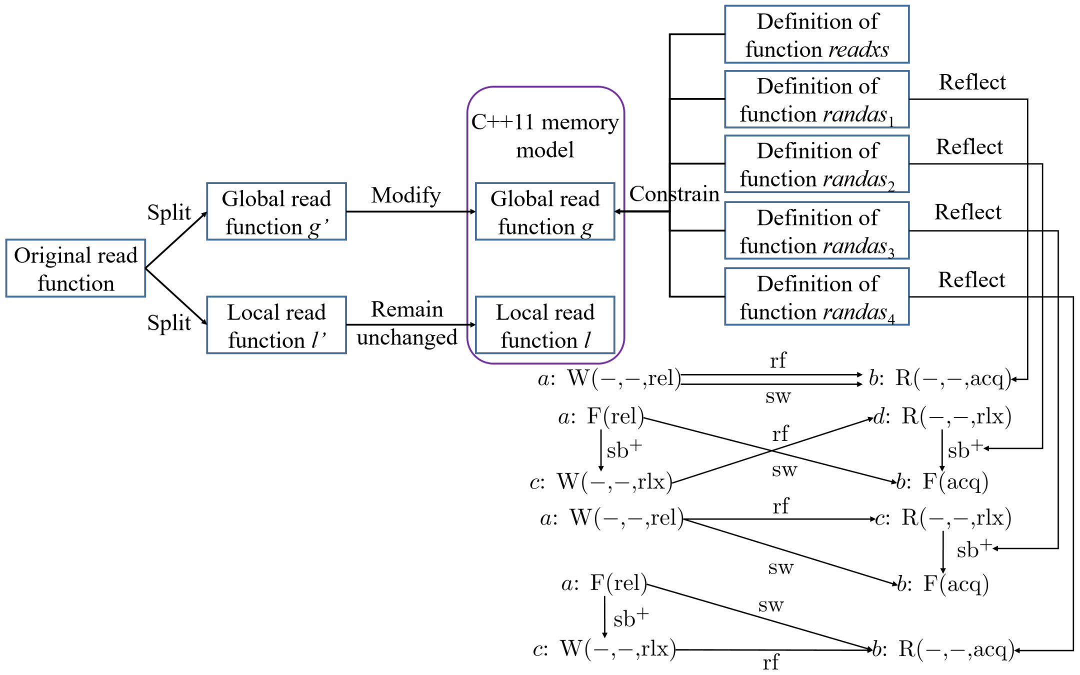

Figure 2: The read mechanism under C++11 (the lower part is adapted from [1]).

Modifications. The interesting features in C++11 will bring in some difficulties to the read operations upon global variables formalized by the function g. Inspired by PS, the principle of g is modified to be: we search the sequence in reverse order until we find a write to x contributed by the thread itself. Then, between the interval, the writes to x produced by the environment can also be read. However, we find that the five above constraints originating from the coherence axiom and the four ways to form synchronization (i.e., the occurrence of release/acquire accesses or release/acquire fence) will narrow the scope of g's execution, presented in Figure 2.

Mechanization. The sequences of any program under C++11 are written to the .xml file with the code: maude -xml-log=relacq.xml. Based on the .xml file, we can get all the possible executions, through the mechanization of the read mechanism with C++ language.

References

[1] M. He. *Reasoning about C11 programs with fences and relaxed atomics.* PhD thesis, Teesside University, Middlesbrough, UK, 2018.

[2] C. A. R. Hoare and He Jifeng. *Unifying theories of programming.* Prentice Hall, 1998.

[3] J. Kang, C. Hur, O. Lahav, V. Vafeiadis, and D. Dreyer. A promising semantics for relaxed-memory concurrency. In *Proc. POPL 2017, Paris, France, January 18-20, 2017*, pages 175–189. ACM, 2017.

[4] L. Xiao, H. Zhu, M. He, and S. Qin. Algebraic semantics for C++11 memory model. In *Proc. COMPSAC 2022, Los Alamitos, CA, USA, June 27 - July 1, 2022*, pages 1–6. IEEE, 2022.