Hindawi Publishing Corporation
Applied Computational Intelligence and Soft Computing
Volume 2014, Article ID 930186, 8 pages
http://dx.doi.org/10.1155/2014/930186

Research Article

Testing Automation of Context-Oriented Programs Using Separation Logic

Mohamed A. El-Zawawy[1,2]

[1] *College of Computer and Information Sciences, Al Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh 11432, Saudi Arabia*
[2] *Department of Mathematics, Faculty of Science, Cairo University, Giza 12613, Egypt*

Correspondence should be addressed to Mohamed A. El-Zawawy; maelzawawy@cu.edu.eg

Received 12 July 2014; Accepted 2 December 2014; Published 29 December 2014

Academic Editor: Baoding Liu

A new approach for programming that enables switching among contexts of commands during program execution is context-oriented programming (COP). This technique is more structured and modular than object-oriented and aspect-oriented programming and hence more flexible. For context-oriented programming, as implemented in COP languages such as ContextJ* and ContextL, this paper introduces accurate operational semantics. The language model of this paper uses Java concepts and is equipped with layer techniques for activation/deactivation of layer contexts. This paper also presents a logical system for COP programs. This logic is necessary for the automation of testing, developing, and validating of partial correctness specifications for COP programs and is an extension of separation logic. A mathematical soundness proof for the logical system against the proposed operational semantics is presented in the paper.

1. Introduction

To support and dynamically control the modularization of crosscutting concerns [1], a new programming style, context-oriented programming (COP) [2], has appeared. COP can be defined as a programming approach that produces software that is dynamically adaptable. COP was invented to treat programming problems when the behavior of the required software changes at runtime depending on the execution conditions. This is not easily achieved by classical approaches of programming. COP was established on languages like Smalltalk [3], Java [4], and JavaScript [5]. While the idea of homogeneous crosscutting is to execute the same source code at the join points of concerns, the idea of heterogeneous crosscutting concerns is to execute different source code at the join points. Heterogeneous crosscutting [1] is the type of crosscutting concerns adopted by COP. Partial functions declarations adapting common functions to their new style are used in COP to implement these crosscutting methodologies.

Main components of COP include (a) layers of variant functions for providing performance alterations and (b) a tool for layer activation/deactavation. A variant function is a function that can be executed after, before, and around the same (variant) function included in another layer. Hence a layer is a group of variant functions.

Layer declarations are used to encapsulate partial function declarations. The semantics of classes and that of crosscutting can be combined at the runtime. COP [2] provides a block statement defining a group of layers to enable runtime layer composition. Determined by the statement block, an execution scope is also provided by COP. In this scope layers are typically combined with the base system. Another statement enables stopping execution of specified layers. The keywords denoting these two statements are *with* and *without* statements [2]. COP appears to be a very convenient technique for encapsulation of homogeneous crosscutting concerns as proved by many applications.

However, so far research did not provide appropriate logic to reason about context-oriented programs. A main concern in COP is the treatment of layers activations/deactivations. Usually, the set of active layers changes from one program point to another. Therefore the efficiency of any logic for

COP strongly depends on using correct layers. In spite of the concept of activating/deactvating a layer is simple, and its logical treatment is a bit complex.

To reason about resources, Reynolds [6] and independently Samin and O'Hearn [7] presented new logic (separation logic). This was based on Burstall's early work. A logical operator (separation conjunction) enabling localizing operation actions on shared resources is among the main contributions of separation logic. Sound form of local reasoning was reached by the separation conjunction which ensures the invariance of resources not included in preconditions.

To model the meanings of COP concepts, this paper introduces operational semantics. Accurate meanings for basics of COP languages are provided by the semantics. Axiomatic semantics for COP is the second contribution of this paper. This semantics is an extension of separation logic to establish and ensure partial correctness specifications for COP programs. Rather than other logic, separation logic seems most suitable for verifying COP. This is so as the concept of context changing is explicitly "treated" by the special operators of separation logic. One of the challenges in this paper is to expose this explicit relationship in the form of formal definitions and inference rules. Judgments of the proposed logic have the form $L \vDash \{P\}S\{Q\}$ meaning that

(i) if layers of L are active, then executing S in a state satisfying P is ensured not to abort,

(ii) if layers of L are active and the execution of S in a state satisfying P ended at some state, then this last state satisfies Q.

Assertions P and Q may reason about the program control locations as well as upon the values of local and global variables. A system of axioms and inference rules constituting the proposed logic is introduced in the paper which presents a partial correctness specification, for example, COP-program, as well. Against the proposed operational semantics, the mathematical soundness of the logical system is shown in this paper.

Contributions of this paper are as follows:

(1) new operational semantics to provide accurate meanings for COP concepts,

(2) novel axiomatic semantics to construct and validate partial correctness specifications for COP programs.

The rest of the paper is organized as follows. Section 2 presents in detail the programming language J-COP and its operational semantics. The assertion language, axioms, and inferences rules constituting the logical system are presented in Section 3 which also presents the mathematical proof for the correctness of the system and an example of its use. Related research is presented in Section 4.

2. Programming Language and Operational Semantics

The syntax and semantics of the programming language (dubbed J-COP) used in this paper are presented in this section. Basic aspects of object-oriented programming like inheritance and subtyping are modeled in J-COP which follows Java syntax for comparable structures. Algorithm 1 presents the syntax of J-COP.

J-Cop is a rich model for COP. The model includes basic language constructs and is extendable directly over well-studied Java features. Although the model is expressive enough to include more language features, it is simple. In addition to typical Java constructs, the model allows layers activation/deactivation, overriding variant functions, and a technique for executing *proceed* and *super*.

We let C denote the typical member of the set of names of classes $\mathscr{C}$ which is included in the set of types (*Types*). In addition to $\mathscr{C}$, *Types* includes function and reference types. The typical element of *Types* is denoted by τ. As it is common in programming, J-COP's functions contain local variables whose scopes are the same as their functions. Also parameters for functions are local variables which are denoted by *LVar*. Instance variables model internal states of classes. We let $IVar_C$ denote the set of instance variables of the class C. The symbols o and v denote typical elements of *LVar* and $IVar_C$, respectively. Sequences of layers activation/deactivation constitute layer expressions (*LayerExpr*) with $le \in LayerExpr$. *FunNames* and *LayerNames* denote the sets of function and layer names, respectively, with $f \in FunNames$ and $l \in LayerNames$.

A class in J-COP includes function definitions and a group of layers containing function definitions. A function is defined via a parameter, a statement, and an expression. The expression models the returned value of the function. A set of classes and a main function (marks the start of program execution) are the components of a J-COP program.

The operational semantics of J-COP presented in this section is defined using state representations and a subtype relation on classes. If C inherits D by definition of C, then C is a *subclass* of D (denoted by $C \ll D$) and D is a *superclass* of C. The reflexive transitive closure of $\ll$ is denoted by $\leq$. States of the operational semantics are presented in Definition 1 where $\mathscr{A}$ denotes an infinite set of memory addresses with $\alpha \in \mathscr{A}$ and $\mathscr{Z}$ is the set of integers.

Definition 1. One has the following:

(1) $\mathscr{D} = \mathscr{Z} \cup \mathscr{A}$;

(2) $Layer_C$ and Fun_C denote the sets of function and layer names of C, respectively;

(3) Stacks $= \{s \mid s : LVar \rightarrow_p \mathscr{Z}\}$;

(4) ObjCont $= \{O_C \mid O_C : IVar_C \rightarrow \mathscr{Z},\ C \in \mathscr{C}\}$;

(5) Heaps $= \{h \mid h : \mathscr{A} \rightarrow_p \text{ObjCont}\}$;

(6) *States* $=$ Stacks $\times$ Heaps.

The set $\mathscr{D}$ includes model values. States of the operational semantics are pairs of a stack and a heap. A special variable (*called this*), to point at the object being executed, is included in the set of local variables. For each class C, we assume two functions F_C and L_C. The map F_C determines for every function $f \in Fun_C$ its components (p_f, S_f, e_f), the parameter variable of the function, the function statement, and the

$$
\begin{aligned}
e \in \text{AExpr} &::= n \mid o \mid e \cdot v \mid e_1 i_{op}\ e_2 \mid (C)e \mid \text{this} \\
b \in \text{BExpr} &::= \text{true} \mid \text{false} \mid e_1 c_{op}\ e_2 \mid b_1 b_{op}\ b_2 \\
le \in \text{LayerExpr} &::= \text{with}\ l \mid \text{without}\ l \mid \epsilon \mid le\ le \\
S \in \text{Stat} &::= e_1 \cdot v := e_2 \mid o_1 := le\ o_2 \cdot f(e) \mid o_1 := o_2 \cdot f(e) \mid \\
&\quad o := super \cdot f(e) \mid o_1 := \text{proceed}\ o_2 \cdot f(e) \mid \\
&\quad o := \text{new}\ C \mid S; S \mid \text{if}\ b\ \text{then}\ S_t\ \text{else}\ S_f \mid \\
&\quad \text{while}\ b\ \text{do}\ S_t \\
\text{funs} \in \text{Fun} &::= f(p)\{S; \text{return}(e);\} \\
\text{layer} \in \text{Lay} &::= \text{Layer}\ l\ \{\text{fun}\} \\
\text{inhrt} \in \text{Inherits} &::= \epsilon \mid \text{inherits}\ C \\
\text{cls} \in \text{Class} &::= \text{class}\ C\ \text{inhrt}\ \{\text{fun}^*\ \text{layer}^*\} \\
\text{prog} \in \text{Prog} &::= \text{cls}^*\ \text{main}()\ \{S\}
\end{aligned}
$$

Algorithm 1: The programming language J-COP.

$$
\begin{aligned}
[C(e)](s,h) &= \begin{cases} undefined, & \text{if } [e](s,h) \in dom(h), h([e](s,h)) = O_D, \text{ and } D \not\leq C; \\ [e](s,h), & \text{otherwise.} \end{cases} \\
[e \cdot v](s,h) &= \begin{cases} O_D(v), & \text{if } h([e](s,h)) = O_D \text{ and } v \in dom(O_D); \\ undefined, & \text{otherwise.} \end{cases}
\end{aligned}
$$

Algorithm 2: Semantics of J-COP expressions.

returned expression of the function. The map I_C determines for every layer $l \in Layer_C$ the parts of the layer's function (f, p_f, S_f, e_f). We also assume a function $\text{SerLay}(L, le, f) = L' = \{l_1, \ldots, l_m\}$ whose inputs are a set of layers, a layer expression, and a function name. SerLay adds (removes) layers activated (deactivated) by *le* to (from) L. Then *SerLay* returns the set of layers in L containing definitions for f.

The semantics of arithmetic and boolean expressions of J-COP is similar to the case of simple imperative language. The cases of $(C)e$ and $e \cdot v$ are shown in Algorithm 2. Some comments on this algorithm are in order. The expression $e \cdot v$ denotes the variable v of the class referenced by e. In line with common OOP concepts, variables of C and that of its ancestors are included in domain of $IVar_C$. Therefore the semantics of $e \cdot v$ is defined only if v is a local variable of the class referenced by e or any of the class's ancestors. The semantics of e (in $e \cdot v$) is supposed to be an address containing a class object. The semantics of $(C)e$ is undefined if e is the address of an object of a class D that is not a descendant of C.

Algorithm 3 presents the semantics of J-COP's statements. The judgments of the semantics have the form $L \dashv S : (s,h) \rightarrow (s',h')$ meaning that executing S in the state (s,h) results in the state (s',h') provided that L is the set of the currently active layers. Some comments on the rules are as follows. The local variable v of the object pointed to by e_1 is updated in the rule $(:=^s_e)$. This amounts to modifying the function O_C in the image of $[e_1](s,h)$ under h. With input e, the rule $(:=^s_{o \cdot f})$ provides semantics for running the function f of the object pointed to by o_2. The semantics of le in $o_1 := le\ o_2 \cdot f(e)$ is to activate/deactivate certain layers. The rule $(:=^s_{l \cdot o \cdot f})$ provides operational semantics for this statement. This rule does the call $\text{SerLay}(L, le, f)$ to add (remove) layers activated (deactivated) via *le* to (from) L. This call then returns the subset of L with definitions for f. The rule then executes the bodies of layer functions sequentially. The execution number i builds on its previous execution by updating o_1 to the value $[e_{i-1}](s_i, h_i)$. The statement $o := \text{super} \cdot f(e)$ running the function f of an ancestor of the current object is given semantics in the rule (sup^s). The rule assumes a function *super* that finds the ancestor of the class pointed to by o_2 such that this ancestor includes a definition for f. Semantics for $o_1 := \text{proceed}\ o_2 \cdot f(e)$ is given in the rule (pro^s). This statement runs all functions f in active layers of the object referenced by o_2.

3. Assertion Language and Logical System

Extended separation logic to cover context-oriented programs is presented in this section. The section introduces the assertion language followed by the logical system's axioms and inference rules. Against the operational semantics presented in the previous section, this section also outlines a formal mathematical proof for the correctness of the logical system. Finally an example of a derivation for a partial correctness specification using the logical system is included in this section as well.

Boolean expressions, classical connectives, first order quantifications, and assertions specific to separation logic are the components of separation logic assertions. The following version of assertions is used in this section.

(i) emp denotes an empty heap.

(ii) $e \mapsto C$ denotes that the heap has a unique memory cell whose address is e and whose content is an instance of the class C.

$$\frac{O_C = h([e_1](s,h)) \quad O'_C = O_C[v \mapsto [e_2](s,h)]}{L \dashv e_1 \cdot v := e_2 : (s,h) \rightarrow (s, h[[e_1](s,h) \mapsto O'_C])} \left(:=^s_e\right)$$

$$\frac{\begin{array}{c} h([o_2](s,h)) = O_C \quad F_C(f) = (p_f, S_f, e_f) \\ L \dashv S_f : (s[\text{this} \mapsto s(o_2), p_f \mapsto [e](s,h)], h) \rightarrow (s'', h'') \end{array}}{L \dashv o_1 := o_2 \cdot f(e) : (s,h) \rightarrow (s''[o_1 \mapsto [e_f](s'', h'')], h'')} \left(:=^s_{o \cdot f}\right)$$

$$\frac{\begin{array}{c} \text{SerLay}(L, le, f) = L' = \{l_1, \ldots, l_m\} \\ h(s(o_2)) = O_C \quad \forall 1 \le i \le m \ (L_C(l_i) = (f, p_i, S_i, e_i)) \\ L' \dashv S_1 : (s[\text{this} \mapsto s(o_2), p_1 \mapsto [e](s,h)], h) \rightarrow (s_2, h_2) \\ \forall i > 1.\ L' \dashv S_i : (s_i[o_1 \mapsto [e_{i-1}](s_i, h_i), \text{this} \mapsto s_i(o_2), \\ p_i \mapsto [e](s_i, h_i)], h_i) \rightarrow (s_{i+1}, h_{i+1}) \end{array}}{L \dashv o_1 := le\ o_2 \cdot f(e) : (s,h) \rightarrow (s_{m+1}[o_1 \mapsto [e_m](s_{m+1}, h_{m+1})], h_{m+1})} \left(:=^s_{l \cdot o \cdot f}\right)$$

$$\frac{\begin{array}{c} h(s(\text{this})) = O_C \quad C \ll E \\ \text{super}(E, f) = D \quad F_D(f) = (p_f, S_f, e_f) \\ L \dashv S_f : (s[p_f \mapsto [e](s,h)], h) \rightarrow (s'', h'') \end{array}}{L \dashv o := \text{super} \cdot f(e) : (s,h) \rightarrow (s''[o \mapsto [e_f](s'', h'')], h'')} \left(\text{sup}^s\right)$$

$$\frac{a \in \mathcal{A} \setminus dom(h)}{L \dashv o := \text{new } C : (s,h) \rightarrow (s[o \mapsto a], h[a \mapsto \emptyset])} \left(\text{new}^s\right)$$

$$\frac{\begin{array}{c} h(s(o_2)) = O_C \\ L' = \{l_1, \ldots, l_m\} \subseteq L. \ (\forall 1 \le i \le m \ (L_C(l_i)) = (f, p_i, S_i, e_i)) \\ L' \dashv S_1 : (s_1[\text{this} \mapsto s(o_2), p_1 \mapsto [e](s_1, h_1)], h_1) \rightarrow (s_2, h_2) \\ \forall i > 1.\ L' \dashv S_i : (s_i[o_1 \mapsto [e_{i-1}](s_i, h_i), \text{this} \mapsto s(o_2), p_i \mapsto [e](s_i, h_i)], h_i) \rightarrow (s_{i+1}, h_{i+1}) \end{array}}{L \dashv o_1 := \text{proceed } o_2 \cdot f(e) : (s,h) \rightarrow (s_{m+1}[o_1 \mapsto [e_m](s_{m+1}, h_{m+1})], h_{m+1})} \left(p^s\right)$$

$$\frac{\begin{array}{l} ([b](s,h) = \text{true} \wedge L \dashv S_t : (s,h) \rightarrow (s', h')) \vee \\ ([b](s,h) = \text{false} \wedge L \dashv S_f : (s,h) \rightarrow (s', h')) \end{array}}{L \dashv \text{if } b \text{ then } S_t \text{ else } S_f : (s,h) \rightarrow (s', h')} \left(\text{if}^s\right)$$

$$\frac{[b](s,h) = \text{false}}{L \dashv \text{while } b \text{ do } S_t : (s,h) \rightarrow (s,h)} \left(\text{whl}^s_1\right) \qquad \frac{\begin{array}{c} L \dashv S_1 : (s,h) \rightarrow (s'', h'') \\ L \dashv S_2 : (s'', h'') \rightarrow (s', h') \end{array}}{L \dashv S_1; S_2 : (s,h) \rightarrow (s', h')} \left(\text{sq}^s\right)$$

$$\frac{\begin{array}{c} [b](s,h) = \text{true} \\ L \dashv S_t : (s,h) \rightarrow (s'', h'') \\ L \dashv \text{while } b \text{ do } S_t : (s'', h'') \rightarrow (s', h') \end{array}}{L \dashv \text{while } b \text{ do } S_t : (s,h) \rightarrow (s', h')} \left(\text{whl}^s_2\right)$$

Algorithm 3: Inference rules of the operational semantics for J-COP constructs.

(iii) this $\mapsto C$ denotes a special case of the previous assertion where $e = \text{this}$.

(iv) $P * Q$ dubbed separating conjunction.

(v) $P -\!\!* Q$ dubbed separating implication.

(vi) $\circledast_{i \in I} P_i$ is a general form of separating conjunction.

Algorithm 4 presents the specific definition of the language of assertions. A group of states are modeled by every assertion via a modeling relation ($(s,h) \vDash P$). The semantics of this relation is that the state (s,h) satisfies the assertion P. Definition 2 formalizes the semantics of the modeling relation.

Definition 2. One has the following:

(i) $(s,h) \vDash \text{fine}(e_1, \ldots, e_n) \overset{\text{def}}{\Leftrightarrow}$ for all i. $[\![e_i]\!](s,h) \neq \text{abort}$;

(ii) $(s,h) \vDash \text{emp} \overset{\text{def}}{\Leftrightarrow} \text{dom}(h) = \emptyset$;

(iii) $(s,h) \vDash e \mapsto C \overset{\text{def}}{\Leftrightarrow} \text{dom}(h) = \{[\![e]\!](s,h)\}$ and $h([\![e]\!](s,h)) = O_C$;

(iv) $(s,h) \vDash P_1 * P_2 \overset{\text{def}}{\Leftrightarrow} \exists h', h''.\ h' \# h'',\ h = h' \cdot h'', (s,h') \vDash P_1$, and $(s,h'') \vDash P_2$;

(v) $(s,h) \vDash P_1 -\!\!* P_2 \overset{\text{def}}{\Leftrightarrow}$ for all $h'((h' \# h$ and $(s,h') \vDash P_1) \Rightarrow (s, h \cdot h') \vDash P_2)$.

The fact that $\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$ is denoted by the expression $h_1 \# h_2$. The union of heaps is denoted by $h_1 \cdot h_2$ and is defined only if $h_1 \# h_2$.

A judgment of the proposed logical system is of the form $L \vDash \{P\} S \{Q\}$ where P and Q are the pre- and postconditions, respectively. The semantics of this judgment is that if layers of L are active and S is executed in a state satisfying P, then the final state will satisfy Q. Definition 3 formalizes the

$$
\begin{aligned}
e \in \text{AExpr} &::= n \mid o \mid e \cdot v \mid e_1 i_{op}\ e_2 \mid (C)e \mid \text{this} \\
b \in \text{BExpr} &::= \text{true} \mid \text{false} \mid e_1 c_{op}\ e_2 \mid b_1 b_{op}\ b_2 \\
P, Q, R, J_1, J_2 \in \text{Asset} &::= b \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q \mid \neg P \mid \forall x.\ P \mid \\
&\quad \exists x.\ P \mid \text{fine}(e) \mid \text{emp} \mid e \mapsto C \mid P * Q \mid \\
&\quad P -\!\!* Q \mid \circledast_{i \in I} P_i.
\end{aligned}
$$

Algorithm 4: The assertion language.

soundness concept of specifications produced by the logical system. Axioms and inference rules of the logical system are introduced in Algorithm 5.

We fix the *abortion* notion for Definition 3. A statement S is *aborting* at (s, h) (denoted by $S : (s, h, L) \rightarrow \text{abort}$) if (a) a state (s', h') such that $S : (s, h) \rightarrow (s', h')$ is not available and (b) S is not stuck in an infinite loop.

Definition 3. A specification $L \dashv \{P\}S\{Q\}$ is sound if, for each (s, h) such that $(s, h) \vDash P$, the following is satisfied:

(1) $\neg(L \vDash S : (s, h) \rightarrow \text{abort})$,

(2) if $L \vDash S : (s, h) \rightarrow (s', h')$, then $(s', h') \vDash Q$.

Some of the rules in Algorithm 5 are certainly noteworthy. In many rules like $(:=^{l}_{e})$ and $(:=^{l}_{o.f})$, the expressions of relevant statements contribute to the preconditions of the rules. This has the advantage of ensuring that the meant memory addresses are indeed allocated and hence evaluations of expressions do not abort. The preconditions of $(:=^{l}_{e})$ and $(:=^{l}_{o.f})$ also ensure that appropriate variables reference objects in heap. The side-conditions of $(:=^{l}_{o.f})$ and $(:=^{l}_{l.o.f})$ consider that executing these statements involves executing certain function bodies. We let $\text{mod}(S, L)$ denote the set of locations modified by S provided that layers of L are active. The symbol $fv(R)$ denotes the set of free variables of R. The rule (share^{l}) enables the composition of specifications having disjoint sets of activated layers if the preconditions of the specifications describe disjoint regions of the heap. The rule (frame^{l}) enables avoiding a frame of the heap, R, that does not affect the statement. This rule also ensures that R is satisfied by the postcondition.

Soundness. The soundness of the logical system is guaranteed by the following theorem.

Theorem 4. *Specifications $L \dashv \{P\}S\{Q\}$ that resulted using the logical system of Algorithm 5 are sound (Definition 3).*

Proof. Structure induction on axioms and inference rules of Algorithm 5 achieves the proof. In the following, basic cases are outlined.

(i) The case of the rule $(:=^{l}_{e})$: in this case

(a) $S = e_1 \cdot v := e_2$,

(b) $P \Leftrightarrow e_1 \mapsto C \wedge \text{fine}(e_1 \cdot v) \wedge \text{fine}(e_2) \wedge Q[e_2/e_1 \cdot v]$.

Suppose that, for state $(s, h), (s, h) \vDash P$. Hence $O_C = h([\![e_1]\!](s, h))$ and $[\![e_2]\!](s, h) \neq \text{abort}$. Therefore $O'_C = O_C[v \mapsto [\![e_2]\!](s, h)]$ is defined and so does $(s, h[[\![e_1]\!](s, h) \mapsto O'_C])$. Hence $\neg(L \vDash S : (s, h) \rightarrow \text{abort})$. Now we suppose $L \vDash S : (s, h) \rightarrow (s', h')$. In this case $(s', h') = (s, h[[\![e_1]\!](s, h) \mapsto O'_C])$. Therefore $(s', h') \vDash Q$ because $(s, h) \vDash Q[e_2/e_1 \cdot v]$. This completes the proof for this case.

(ii) The case of the rule $(*:=^{l}_{o.f})$: in this case

(a) $S = o_1 := o_2 \cdot f(e)$,

(b) $F_C(f) = (p_f, S_f, e_f)$,

(c) $L \dashv \{R[o_2/\text{this}, e/p_f]\}\ S_f\ \{Q[e_f/o_1]\}$,

(d) $L \dashv S_f : (s[\text{this} \mapsto s(o_2), p_f \mapsto [\![e]\!](s, h)], h) \rightarrow (s'', h'')$,

(e) $P \Leftrightarrow o_2 \mapsto C \wedge R \wedge \text{fine}(e, e_f)$.

Suppose that, for state (s, h), $(s, h) \vDash P$. This implies $h([\![o_2]\!](s, h)) = O_C$. The condition $\text{fine}(e, e_f)$ implies $(s[\text{this} \mapsto s(o_2), p_f \mapsto [\![e]\!](s, h)], h)$ is defined. Therefore $(s[\text{this} \mapsto s(o_2), p_f \mapsto [\![e]\!](s, h)], h) \vDash R[o_2/\text{this}, e/p_f]$. By induction hypothesis S_f does not abort at $(s[\text{this} \mapsto s(o_2), p_f \mapsto [\![e]\!](s, h)], h)$ and therefore $o_1 := o_2 \cdot f(e)$ does not abort at (s, h). Now we suppose that $o_1 := o_2 \cdot f(e) : (s, h) \rightarrow (s', h')$. By $(:=^{s}_{o.f})$, $(s', h') = (s''[o_1 \mapsto [\![e_f]\!](s'', h'')], h'')$. Also by induction hypothesis $(s'', h'') \vDash Q[e_f/o_1]$ implying $(s', h') \vDash Q$. This completes the proof for this case.

(iii) The case of the rule $(*:=^{l}_{l.o.f})$: in this case

(a) $S = o_1 := le\ o_2 \cdot f(e)$,

(b) $\text{SerLay}(L, le, f) = L' = \{l_1, \ldots, l_m\}$, for all $1 \leq i \leq m(L_C(l_i) = (f, p_i, S_i, e_i))$,

(c) $L' \dashv S_1 : (s[\text{this} \mapsto s(o_2), p_1 \mapsto [\![e]\!](s, h)], h) \rightarrow (s_2, h_2)$,

(d) for all $i > 1$. $L' \dashv S_i : (s_i[o_1 \mapsto [\![e_{i-1}]\!](s_i, h_i), \text{this} \mapsto s_i(o_2), p_i \mapsto [\![e]\!](s_i, h_i)], h_i) \rightarrow (s_{i+1}, h_{i+1})$,

(e) $L' \dashv \{R[o_2/\text{this}, e/p_1]\}\ S_1\ \{Q_1[e_1/o_1]\}$,

(f) for all $i > 1$. $L' \dashv \{Q_{i-1}[o_2/\text{this}, e/p_i]\}\ S_i\ \{Q_i[e_i/o_1]\}$,

(g) $P \Leftrightarrow o_2 \mapsto C \wedge R \wedge \text{fine}(e, e_1, \ldots, e_m)$.

Suppose that, for state (s, h), $(s, h) \vDash P$. This implies $h([\![o_2]\!](s, h)) = O_C$. The condition $\text{fine}(e, e_1, \ldots, e_m)$ implies $(s[\text{this} \mapsto s(o_2), p_1 \mapsto [\![e]\!](s, h)], h)$ is defined. Therefore $(s[\text{this} \mapsto s(o_2), p_1 \mapsto [\![e]\!](s, h)], h) \vDash R[o_2/\text{this}, e/p_1]$. By induction hypothesis S_1 does not abort at $(s[\text{this} \mapsto s(o_2), p_1 \mapsto [\![e]\!](s, h)], h)$.

$$\frac{}{L \dashv \{e_1 \mapsto C \wedge \mathrm{fine}(e_1 \cdot v) \wedge \mathrm{fine}(e_2) \wedge Q[e_2/e_1 \cdot v]\}\ e_1 \cdot v := e_2 \{Q\}}\ (:=_e^l)$$

$$\frac{F_C(f) = (p_f, S_f, e_f) \quad L \dashv \{R[o_2/\mathrm{this}, e/p_f]\}\ S_f\ \{Q[e_f/o_1]\}}{L \dashv \{o_2 \mapsto C \wedge R \wedge \mathrm{fine}(e, e_f)\}\ o_1 := o_2 \cdot f(e)\ \{Q\}}\ (:=_{o\cdot f}^l)$$

$$\frac{\mathrm{SerLay}(L, le, f) = L' = \{l_1, \ldots, l_m\} \quad L' \dashv \{R[o_2/\mathrm{this}, e/p_1]\}\ S_1\ \{Q_1[e_1/o_1]\} \quad \forall i > 1.\ L' \dashv \{Q_{i-1}[o_2/\mathrm{this}, e/p_i]\}\ S_i\ \{Q_i[e_i/o_1]\}}{L \dashv \{o_2 \mapsto C \wedge R \wedge \mathrm{fine}(e, e_1, \ldots, e_m)\}\ o_1 := le\ o_2 \cdot f(e)\ \{Q_m\}}\ (:=_{l\cdot o\cdot f}^l)$$

$$\frac{\mathrm{super}(E, f) = D \quad F_D(f) = (p_f, S_f, e_f) \quad L \dashv \{R[e/p_f]\}\ S_f\ \{Q[e_f/o]\}}{L \dashv \{\mathrm{this} \mapsto C \wedge C \ll E \wedge R \wedge \mathrm{fine}(e, e_f)\}\ o := \mathrm{super} \cdot f(e)\ \{Q\}}\ (\mathrm{sup}^l)$$

$$\frac{}{L \dashv \{\mathrm{emp}\}\ o := \mathrm{new}\ C\ \{o \mapsto C\}}\ (\mathrm{new}^l)$$

$$\frac{L' = \{l_1, \ldots, l_m\} \subseteq L.\ (\forall 1 \le i \le m\ (L_C(l_i)) = (f, p_i, S_i, e_i)) \quad L' \dashv \{R[o_2/\mathrm{this}, e/p_1]\}\ S_1\ \{Q_1[e_1/o_1]\} \quad \forall i > 1.\ L' \dashv \{Q_{i-1}[o_2/\mathrm{this}, e/p_i]\}\ S_i\ \{Q_i[e_i/o_1]\}}{L \dashv \{o_2 \mapsto C \wedge R\}\ o_1 := \mathrm{proceed}\ o_2 \cdot f(e)\ \{Q_m\}}\ (\mathrm{pro}^l)$$

$$\frac{L \dashv \{P\}\ S_1\ \{R\} \quad L \dashv \{R\}\ S_2\ \{Q\}}{L \dashv \{P\}\ S_1; S_2\ \{Q\}}\ (\mathrm{Seq}^l) \qquad \frac{L \dashv \{P\}\ S\ \{Q\} \quad L \dashv \{P'\}\ S\ \{Q'\}}{L \dashv \{P \vee P'\}\ S\ \{Q \vee Q'\}}\ (\mathrm{Disj}^l)$$

$$\frac{L \dashv \{P \wedge b\}\ S_t\ \{Q\} \quad L \dashv \{P \wedge \neg(b)\}\ S_f\ \{Q\}}{L \dashv \{P\}\ \mathrm{if}\ b\ \mathrm{then}\ S_t\ \mathrm{else}\ S_f\ \{Q\}}\ (\mathrm{if}^l)$$

$$\frac{L \dashv \{\mathrm{fine}(b) \wedge P\}\ S_t\ \{\mathrm{fine}(b) \wedge P\}}{L \dashv \{\mathrm{fine}(b) \wedge P\}\ \mathrm{while}\ b\ \mathrm{do}\ S_t\ \{\neg(b) \wedge P\}}\ (\mathrm{while}^l)$$

$$\frac{L \dashv \{P\}\ S\ \{Q\} \quad P' \Longrightarrow P \quad Q \Longrightarrow Q'}{L \dashv \{P'\}\ S\ \{Q'\}}\ (\mathrm{seq}^l) \qquad \frac{L_1 \dashv \{P\}\ S\ \{Q\} \quad L_2 \dashv \{P'\}\ S\ \{Q'\}}{L_1 \cup L_2 \dashv \{P \wedge P'\}\ S\ \{Q \wedge Q'\}}\ (\mathrm{Conj}^l)$$

$$\frac{L_1 \dashv \{P\}\ S\ \{Q\} \quad L_2 \dashv \{R\}\ S\ \{R\} \quad L_1 \cap L_2 = \emptyset}{L_1 \cup L_2 \dashv \{P * R\}\ S\ \{Q * R\}}\ (\mathrm{share}^l) \qquad \frac{L \dashv \{P\}\ S\ \{Q\} \quad fv(R) \cap \mathrm{mod}(S, L) = \emptyset}{L \dashv \{P * R\}\ S\ \{Q * R\}}\ (\mathrm{frame}^l)$$

$$\frac{\{l\} \dashv \{P\}\ S\ \{Q\} \quad L \setminus \{l\} \dashv \{J_1\}\ S\ \{J_2\}}{L \dashv \{P \wedge J_1\}\ S\ \{Q \vee J_2\}}\ (\mathrm{glob}^l) \qquad \frac{L \dashv \{P\}\ S\ \{Q\} \quad x \notin fv(S)}{L \dashv \{\exists x.\ P\}\ S\ \{\exists x.\ Q\}}\ (\mathrm{Ex}^l)$$

Algorithm 5: The inference rules of the proposed logical system.

Now we suppose that $S_1 : (s, h) \to (s_2, h_2)$. By induction hypothesis $(s_2, h_2) \vDash Q[e_1/o_1]$ implying $(s_2[o_1 \mapsto [\![e_1]\!](s_2, h_2), \mathrm{this} \mapsto s_2(o_2), p_2 \mapsto [\![e]\!](s_2, h_2)], h_2) \vDash Q_1[o_2/\mathrm{this}, e/p_2]$. Therefore by induction hypothesis S_2 does not abort at $(s_2[o_1 \mapsto [\![e_1]\!](s_2, h_2), \mathrm{this} \mapsto s_2(o_2), p_2 \mapsto [\![e]\!](s_2, h_2)], h_2)$. Now we suppose that $S_2 : (s_2[o_1 \mapsto [\![e_1]\!](s_2, h_2), \mathrm{this} \mapsto s_2(o_2), p_2 \mapsto [\![e]\!](s_2, h_2)], h_2) \to (s_3, h_3)$. By induction hypothesis $(s_3, h_3) \vDash Q[e_2/o_1]$ implying $(s_3[o_1 \mapsto [\![e_2]\!](s_3, h_3), \mathrm{this} \mapsto s_3(o_2), p_3 \mapsto [\![e]\!](s_3, h_3)], h_3) \vDash Q_2[o_2/\mathrm{this}, e/p_3]$. Therefore by induction hypothesis S_3 does not abort at $(s_3[o_1 \mapsto [\![e_2]\!](s_3, h_3), \mathrm{this} \mapsto s_3(o_2), p_3 \mapsto [\![e]\!](s_3, h_3)], h_3)$. Hence a simple induction on m completes the proof for this case.

(iv) The case of the rule (sup^l): in this case

(a) $S = o := \mathrm{super} \cdot f(e)$,
(b) $\mathrm{super}(E, f) = D$,
(c) $F_D(f) = (p_f, S_f, e_f)$,
(d) $L \dashv S_f : (s[p_f \mapsto [\![e]\!](s, h)], h) \to (s'', h'')$,
(e) $L \dashv \{R[e/p_f]\}\ S_f\ \{Q[e_f/o]\}$,
(f) $P \Leftrightarrow \mathrm{this} \mapsto C \wedge C \ll E \wedge R \wedge \mathrm{fine}(e, e_f)$.

Suppose that, for state (s,h), $(s,h) \vDash P$. This implies $h([\![\text{this}]\!](s,h)) = O_C$. The condition fine$(e, e_f)$ implies $(s[p_f \mapsto [\![e]\!](s,h)], h)$ is defined. Therefore $(s[\text{this} \mapsto s(o_2), p_f \mapsto [\![e]\!](s,h)], h) \vDash R[e/p_f]$. By induction hypothesis S_f does not abort at $(s[p_f \mapsto [\![e]\!](s,h)], h)$ and therefore $o := \text{super} \cdot f(e)$ does not abort at (s,h). Now we suppose that $o_1 := o_2 \cdot f(e) : (s,h) \to (s',h')$. By $(:=^{s}_{o\cdot f})$, $(s',h') = (s''[o \mapsto [\![e_f]\!](s'',h'')], h'')$. Also by induction hypothesis $(s'',h'') \vDash Q[e_f/o]$ implying $(s',h') \vDash Q$. This completes the proof for this case.

(v) The case of the rule (pro^l): it is similar to the case of $(*:=^{l}_{l\cdot o\cdot f})$. □

4. Discussion

Related Work. The most related logic to our work is that of object-oriented programs. Specifically for Java, a huge literature on specification proof techniques for object-oriented languages exists. For example, for annotated Java programs in JML (Java modeling language), [8] presents calculus for weakest precondition. An introduction to applications and tools of JML is presented in [9]. Via annotating Java source files, JML enables determining Java interfaces and classes. Considering abnormal termination resulting from failures of Java programs, [10] presents an extension to Hoare logic. For theses failures, transformational techniques do not work.

Notably, dynamically growing reference-structures are involved in object-oriented programs. This results in the aliasing problem in OOP languages. Much logic treating aliasing does exist. One of these techniques is presented in [11] and reasons about linked lists which are treated using separation logic [6]. Via including global stores in the assertion language, [12] presents Hoare logic for OOP languages. However the use of this model for global storing is proved to be a potential cause of incompleteness. Classes encapsulation in an OOP language equipped with subtyping and pointers is guaranteed by aliasing restrictions in [13].

The work in [14] is an example of logic achieving modular verification for OOP and focusing on corresponding methodology and object invariants. Modular verification of complex object structures was treated in [15] using an invariant class. Sound proof systems producing restricted formal justifications for OOP languages were introduced in [16, 17]. Producing a complete logical system for OOP languages is a very complex problem due to complex features of OOP languages.

Interestingly, none of the techniques mentioned above treat context-oriented programs. This adds to the value of the work presented in the current paper; apparently there is no research to generalize separation logic to cover context-oriented programs.

Operational semantics and type systems are presented in [18, 19] for checking and modeling context-oriented programs. The language model of [19] is structural and very similar to the model of the current paper. The language model of [18] is functional. Type systems introduced in both of these papers prevent execution of faculty functions by *proceed*. The operational semantics of the current paper is much simpler and powerful than semantics of [18, 19].

Utilizing concepts of delegation based calculus, [20] presents operational semantics for a COP language, namely, *cj*. For COP concepts as introduced in ContextJ* and ContextL, [21] introduces syntax-based semantics. Compared to the work mentioned above, a logical system that stops COP program from getting stuck is presented in the current paper. To model context-dependent behavior of COP programs, the proposed logical system is based on general calculi.

Future Work. Extending the language model of the current paper to allow executing candidate functions of *proceed* on priority bases is a direction for a future work. Extending the language to include layer inheritance and layer dependency is another direction for future work. Doing so allows one layer to assume the existence of another layer and allows expressing the assumption that two layers are not active at the same time.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

References

[1] G. Kiczales, J. Lamping, A. Mendhekar et al., "Aspect-oriented programming," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP '97)*, pp. 220–242, Jyväskylä, Finland, June 1997.

[2] R. Hirschfeld, P. Costanza, and O. Nierstrasz, "Context-oriented programming," *The Journal of Object Technology*, vol. 7, no. 3, pp. 125–151, 2008.

[3] W. Golubski and W.-M. Lippe, "A complete semantics for SMALLTALK-80," *Computer Languages*, vol. 21, no. 2, pp. 67–79, 1995.

[4] M. Campione, K. Walrath, and A. Huml, *The Java Tutorial: A Short Course on the Basics*, Addison-Wesley Longman Publishing, Boston, Mass, USA, 3rd edition, 2000.

[5] D. Flanagan, *JavaScript—Pocket Reference: Activate Your Web Pages*, O'Reilly, 3rd edition, 2012.

[6] J. C. Reynolds, "Separation logic: a logic for shared mutable data structures," in *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS '02)*, pp. 55–74, July 2002.

[7] S. S. Samin and P. W. O'Hearn, "BI as an assertion language for mutable data structures," in *Principles of Programming Languages*, C. Hankin and D. Schmidt, Eds., pp. 14–26, ACM, 2001.

[8] B. Jacobs, "Weakest pre-condition reasoning for Java programs with JML annotations," *Journal of Logic and Algebraic Programming*, vol. 58, no. 1-2, pp. 61–88, 2004.

[9] L. Burdy, Y. Cheon, D. R. Cok et al., "An overview of JML tools and applications," *International Journal on Software Tools for Technology Transfer*, vol. 7, no. 3, pp. 212–232, 2005.

[10] M. Huisman and B. Jacobs, "Java program verification via a hoare logic with abrupt termination," in *Fundamental Approaches to Software Engineering*, T. S. E. Maibaum, Ed., vol. 1783 of *Lecture Notes in Computer Science*, pp. 284–303, Springer, Berlin, Germany, 2000.

[11] J. M. Morris, "A general axiom of assignment," in *Theoretical Foundations of Programming Methodology*, pp. 25–34, Springer, Berlin, Germany, 1982.

[12] N. Dershowitz, Ed., *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, vol. 2772 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2003.

[13] A. Banerjee and D. A. Naumann, "Ownership confinement ensures representation independence for object-oriented programs," *Journal of the ACM*, vol. 52, no. 6, pp. 894–960, 2005.

[14] M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, and K. R. M. Leino, "Boogie: a modular reusable verifier for object-oriented programs," in *Formal Methods for Components and Objects*, F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, Eds., vol. 4111 of *Lecture Notes in Computer Science*, pp. 364–387, Springer, Berlin, Germany, 2006.

[15] P. Müller, A. Poetzsch-Heffter, and G. T. Leavens, "Modular invariants for layered object structures," *Science of Computer Programming*, vol. 62, no. 3, pp. 253–286, 2006.

[16] D. von Oheimb, "Hoare logic for Java in Isabelle/hol," *Concurrency Computation Practice and Experience*, vol. 13, no. 13, pp. 1173–1214, 2001.

[17] G. Klein and T. Nipkow, "A machine-checked model for a Java-like language, virtual machine, and compiler," *ACM Transactions on Programming Languages and Systems*, vol. 28, no. 4, pp. 619–695, 2006.

[18] R. Hirschfeld, A. Igarashi, and H. Masuhara, "Contextfj: a minimal core calculus for context-oriented programming," in *Proceedings of the 10th International Workshop on Foundations of Aspect-Oriented Languages (FOAL '11)*, pp. 19–23, ACM, March 2011.

[19] M. A. El-Zawawy and E. A. Aleisa, "A new model for context-oriented programs," *Life Science Journal*, vol. 10, no. 2, pp. 2515–2523, 2013.

[20] H. Schippers, D. Janssens, M. Haupt, and R. Hirschfeld, "Delegation-based semantics for modularizing crosscutting concerns," in *Proceedings of the 23rd ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '08)*, pp. 525–542, Nashville, Tenn, USA, October 2008.

[21] D. Clarke and I. Sergey, "A semantics for context-oriented programming with layers," in *Proceedings of the International Workshop on Context-Oriented Programming (COP '09)*, ACM, Genova, Italy, July 2009.