Completeness of Verification System with Separation Logic for Recursive Procedures

Mahmudul Faisal Al Ameen

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences
SOKENDAI (The Graduate University for Advanced Studies)

Completeness of Verification System with Separation Logic for Recursive Procedures

Mahmudul Faisal Al Ameen

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences
SOKENDAI (The Graduate University for Advanced Studies)
Tokyo, Japan

September 2016

A dissertation submitted to
the Department of Informatics
School of Multidisciplinary Sciences
SOKENDAI (The Graduate University for Advanced Studies)
in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy

Review Committee

Makoto TATSUTA	National Institute of Informatics, SOKENDAI
Zhenjiang HU	National Institute of Informatics, SOKENDAI
Makoto KANAZAWA	National Institute of Informatics, SOKENDAI
Shin Nakajima	National Institute of Informatics, SOKENDAI
Yukiyoshi Kameyama	University of Tsukuba

Thesis advisor: Makoto Tatsuta Mahmudul Faisal Al Ameen

SOKENDAI (The Graduate University for Advanced Studies)

Abstract

Completeness of Verification System with Separation Logic for Recursive Procedures

The contributions of this dissertation are two results; the first result gives a new complete Hoare's logic system for recursive procedures, and the second result proves the completeness of the verification system based on Hoare's logic and separation logic for recursive procedures. The first result is a complete verification system for reasoning about *WHILE* programs with recursive procedures that can be extended to separation logic. To obtain it, this work introduces two new inference rules, shows derivability of an inference rule and removes other redundant inference rules and an unsound axiom for showing completeness. The second result is a complete verification system, which is an extension of Hoare's logic and separation logic for mutual recursive procedures. To obtain the second result, the language of *WHILE* programs with recursive procedures is extended with commands to allocate, access, mutate and deallocate shared resources, and the logical system from the first result is extended with the backward reasoning rules of Hoare's logic and separation logic. Moreover, it is shown that the assertion language of separation logic is expressive relative to the programs. It also introduces a novel expression that is used to describe the complete information of a given state in a precondition. In addition, this work uses the necessary and sufficient precondition of a program for the abort-free execution, which enables to utilize the strongest postconditions.

I would like to dedicate this thesis to my parents for their love, sacrifice and support all through my life.

Acknowledgments

I wish to express my deepest gratitude to my advisor, Prof. Makoto Tatsuta, for giving me the opportunity, pre-admission supports and recommendations to study in SOKENDAI with NII scholarship. His guidance with extreme patience is the key driving force for my today's success. His support not only encouraged but also nourished me this last six years. Without his sincere efforts, I could not be in today's position. His lessons helped me realizing the rigid nature of mathematics and logic.

I am very grateful to my co-supervisor Asst. Prof. Makoto Kanazawa for being one of the most important supporting force for my work. His suggestions at the logic seminars have been a very good basis for elaborating and structuring my research; he gave me enough feedback to mature my ideas while always pointing me interesting directions.

I would like to thank Prof. Zhenjiang Hu for providing valuable remarks, allowing me to improve the papers drafts and clarify my arguments. He has shown me a very important horizon of program composition that enabled me to realize programs as mathematical objects. His care and support especially kept me optimistic in my hard times.

Valuable review and suggestions made by Prof. Shin Nakajima of SOKENDAI and Prof. Yukiyoshi Kameyama of University of Tsukuba helped me to enhance and fine tuning the dissertation. I am very grateful them.

I am grateful to Prof. Kazushige Terui for his suggestions and support in the early stage of my study.

I would like to express my sincere appreciation to Dr. Daisuke Kimura for giving me a long time support as an elder brother. I am indebted to Dr. Takayuki Koai for his sincere most effort to make my life easy in Japan. In different occasions, their academic tutoring, as well as mental supports, are never to be forgotten. Their explanation of several basic and advanced topics often made me understand difficult mathematical topics in an easy way. Special thanks to Dr. Kazuhiro Inaba for an important criticism to show a gap in one of my earlier solution. Dr. Kazuyuki Asada's advice and Mr. Toshihiko Uchida's friendly support also helped me to reach in this position. I am very grateful to all of them. Moreover, I thank to Mr. Koike Atsushi for sharing his LaTeX class file, based on which the class file for this dissertation is prepared.

I thank National Institute of Informatics for providing me the necessary financial and resource support. Further, I also thank NII officials at the student support section for their administrative assistances.

I owe to Prof. Dr. Shorif Uddin for driving me and directly helping me to apply to NII for the doctoral study. I also thank Dr. Zahed Hasan and Mr. Enayetur Rahman for their contributions in my life and research career.

Finally, I thank my parents for their love, sacrifices, long patience and endless encouragements. I thank my wife for her invaluable and direct care and support in the final year. I also thank my sister, relatives and friends for their great mental support that helped me to handle the challenging research situations.

Contents

1
Introduction

1.1 Motivation

It is widely accepted that a program is needed to be verified to ensure that it is correct. A correct program guarantees to perform the given task as expected. It is very important to ensure the safety of the mission-critical, medical, spacecraft, nuclear reactor, financial, genetic-engineering and simulator programs. Moreover, everyone desires bug-free programs.

Formal verification is intended to deliver programs which are completely free of bugs or defects. It verifies the source code of a program statically. So formal verification does not depend on the execution of a program. The time required to verify a program depends on neither its runtime and memory complexity nor the magnitude of its inputs. A program is required to be verified only once since it does not depend on test cases. Hence, formal verification of programs is important to save both time and expenses commercially and for its supremacy theoretically. Among formal verification approaches, model checking and Hoare's logic are prominent. Model checking computes whether a model satisfies a given specification, whereas Hoare's logic shows it for all models by provability [12].

Since it was proposed by Hoare [11], numerous works on Hoare's logic have been done [3, 6, 8, 10, 14]. Several extensions have also been proposed [3], among which some attempted to verify programs that access heap or shared resources. However, until the twenty-first century begins, very few of them were simple enough to use. On the other hand, since the development of programming languages like C and C++, the usage of pointers in programs (which are called pointer programs) gained much popularity for their ability to use shared memory and other resources directly and for faster execution. This ability also causes crashed programs for some reasons because it is difficult to keep track of each memory operation. It may lead unsafe heap operation. A program crash occurs when the program tries to access a memory cell that has already been deallocated before or when a memory cell is accessed before its allocation. So apparently it became necessary to have an extension of Hoare's logic that can verify such pointer programs. In 2002, Reynolds proposed separation logic [18]. It was a breakthrough to achieve the ability to verify pointer programs. Especially it can guarantee safe heap operations of programs. Although recently we can find several works on separation logic [4, 13] and its extensions and applications [5, 15, 16], there are few works found to show their completeness [19]. *Tatsuta et al.* [19] show the completeness of the separation logic for pointer programs which is introduced in [18]. In this paper, we will show the completeness of an extended logical system. Our logical system is intended to verify pointer programs with mutual recursive procedures. Among several versions of the same inference rule, Reynolds offered in [18] for separation logic, a concise set of backward reasoning rules has been chosen in [19]. The later work in [19] also offers rigorous mathematical discussions. The problems regarding the completeness of Hoare's logic, the concept of relative completeness, completeness of Hoare's logic with recursive procedures and many other important topics have been discussed in detail in [3]. Our work begins with [19] and [3].

In modern days, programs are written in segments with procedures, which make the programs shorter in size and logically structured, and increase the reusability of code. So it is important to use procedures and heap operations (use of shared mutable resources) both in a single program. The parameter mechanism is an important part of a procedure, and it enhances the flexibility in programming. However, theoretically, the parameterless procedures are simpler to analyze, and it is much easier to extend it with parameters. Moreover, there are different kinds of parameter mechanism such as call-by-value, call-by-name, and call-by-reference. So verification of pointer programs with parameterless procedures is a significant starting point of verification of programs with different parameter mechanisms. Therefore, it is important to achieve a sound and complete verification system for pointer programs with parameterless procedures first so that it can be extended to different parameter mechanisms later. It is the main motivation of our work.

1.2 Main Contribution

Our goal is to give a relatively complete logical system that can be used for reasoning about pointer programs with mutual recursive procedures. A logical system for software verification is called complete if every true judgment can be derived using that system. It ensures the strength of our system so that no further development is necessary for the logical system. If all true asserted programs are provable in Hoare's system where all true assertions are provided, we call it a relatively complete system. We will show the relative completeness of our system. A language is expressive if the weakest precondition can be defined in the language. We will also show that our language of specification is expressive for our programs. Relative completeness is discussed vastly in [3, 8]. In this paper, relative completeness is sometimes paraphrased as completeness when it is not ambiguous.

The main contributions of our paper are as follows:

(1) A new complete logical system for Hoare's logic for recursive procedures [1].

(2) A new logical system for verification of pointer programs and recursive procedures [2].

(3) Proving the soundness and the completeness theorems.

(4) Proving that our assertion language is expressive for our programs.

(5) Discussing soundness and admissibility of the frame rules and the conjunction rule in our system.

We know that Hoare's logic with recursive procedures is complete [3]. We also know that Hoare's logic with separation logic is complete [19]. But we do not know if Hoare's logic with separation logic for recursive procedures is complete.

To achieve our contributions, we will first construct our logical system by combining the axioms and inference rules of [3] and [19]. Then we will prove the expressiveness by coding the states in a similar way to [19]. At last, we will follow a similar strategy in [3] to prove the completeness.

Although one may feel it easy to combine these two logical systems to achieve such a complete system, in reality, it is not the case. Now we will discuss some challenges we face to prove its relative completeness.

(1) The axiom (Axiom 9: Invariance Axiom) is defined in [3] by $\overline{\{A\}P\{A\}}$ where free variables of P and A are mutually exclusive, P is a *WHILE* program with recursive procedures, and A is an assertion in Hoare's logic. It is an essential axiom to show completeness of Hoare's logic but it is not sound in separation logic.

(2) In the completeness proof of the extension of Hoare's logic for the recursive procedures in [3], the expression $\overrightarrow{x} = \overrightarrow{z}$ ($\overrightarrow{x}$ are all program variables, and $\overrightarrow{z}$ are fresh) is used to describe the complete information of a given state in a precondition. A state in Hoare's logic is only a store, which is a mapping from the set of variables to the set of natural numbers. In separation logic, a state is a pair of a store and a heap. So the same expression cannot be used for a similar purpose for a heap because a store information may contain variables $x_1, \ldots, x_m$ which are assigned $z_1, \ldots, z_m$ respectively, while a heap information consists of the set of the physical addresses only in the heap and their corresponding values. The vector notation cannot express the general information of the size of the heap and its changes because of allocation and deallocation of memory cells.

(3) Another challenge is to utilize the strongest postcondition of a precondition and a program. In case a program aborts in a state for which the precondition is valid, the strongest postcondition of the precondition and the program does not exist. But utilizing the strongest postcondition is necessary for completeness proof because the completeness proof of [3] depends on it.

Now it is necessary to solve these obstacles for the proof of the completeness of our system. That is why it is quite challenging to solve the completeness theorem which is our principal goal.

The solutions to the challenges stated above are as follows:

(1) We will give an inference rule (INV-CONJ) as an alternative to the axiom (AXIOM 9: INVARIANCE AXIOM) in [3]. It will accept a pure assertion which does not have a variable common to the program. We will also give an inference rule (EXISTS) that is analogous to the existential introduction rule in the first-order predicate calculus. We will show that the inference rule (RULE 10: SUBSTITUTION RULE I) in [3] is derivable in our system. Since the inference rules (RULE 11: SUBSTITUTION RULE II) and (RULE 12: CONJUNCTION RULE) in [3] are redundant in our system, we will remove them. It gives us the new complete system for Hoare's logic for mutual recursive procedures. We will extend this system with the inference rules in [19] to give the verification system for pointer programs with mutual recursive procedures. As a result, the set of our axioms and inference rules will be quite different from the union of those of [3] and [19].

(2) We will give an appropriate assertion to describe the complete information of a given state in a precondition. Beside the expression $\overrightarrow{x} = \overrightarrow{z}$ for the store information, we will additionally use the expression $\mathrm{Heap}(x_h)$ for the heap information, where x_h keeps a natural number that is obtained by a coding of the current heap.

(3) For pointer programs, it is difficult to utilize the strongest postcondition because it is impossible to assert a postcondition for A and P where P may abort in a state for which A is true. We use $\{A\}P\{\mathrm{True}\}$ as the abort-free condition of A and P. For the existence of the strongest postcondition, it is necessary for $\{A\}P\{\mathrm{True}\}$ to be true. We will give the necessary and sufficient precondition $W_{P,\mathrm{True}}(\overrightarrow{x})$ for the fact

that the program P will never abort.

1.3 Outline of This Paper

Our background will be presented in Chapter 2. A new complete Hoare's logic for recursive procedures will be given in Chapter 3. It will be extended to a complete system with separation logic in the next chapter. We will define our languages, semantics and the logical system in Chapter 4. In Chapter 5, we will prove the soundness, expressiveness and completeness. Admissibility of some important inference rules in our system will be discussed in Chapter 6. We will conclude in Chapter 7.

2
Background

Hoare introduced an axiomatic method, Hoare's logic, to prove the correctness of programs in 1969 [11]. Floyd's intermediate assertion method was behind the approach of Hoare's logic. Besides its great influence in designing and verifying programs, it has also been used to define the semantics of programming languages.

While Hoare's logic is sound, it is not complete since Peano arithmetic is undecidable. If Hoare's logic contains a proof system of Peano arithmetic, the Hoare's logic becomes undecidable. Cook indicated a way to overcome these difficulties by defining the notion of completeness in 1978 [8]. If there exists an assertion in $\mathcal{L}$ that defines the strongest postcondition of $A \in \mathcal{L}$ and $P \in \mathcal{P}$, $\mathcal{L}$ is said to be expressive relative to $\mathcal{P}$. A proof system for $\mathcal{P}$ and $\mathcal{L}$ is complete in the sense of Cook if $\mathcal{L}$ is expressive relative to $\mathcal{P}$ and all true assertions are given. Cook also extended Hoare's logic to nonrecursive procedures and proved its completeness in the above sense. Gorelick [9] extended Cook's work to recursive procedures.

Among many works which extended the approach of Hoare to prove a program correct, some were not useful or difficult to use. In 1981, Apt presented a survey of various results concerning the approach of Hoare in [3]. His work emphasized mainly on the soundness and completeness issues. He first presented the proof system for *WHILE* programs along with its soundness, expressiveness, and completeness in the sense of Cook. He then presented the work of Gorelick, the extension of Hoare's logic to recursive procedures. He also presented other extensions such as local variable declarations and procedures with parameters with corresponding soundness and completeness.

2.1 Hoare's Logic for Recursive Procedures

In this section, we will discuss the verification system for *WHILE* programs with recursive procedures given in [3]. Here we will present the language of the *WHILE* programs with recursive procedures and the assertions, their semantics, a logical system to reason about the programs and the completeness proof. We will extend this proof to pointer programs with recursive procedures later.

2.1.1 Language

The language of assertion in [3] is the first order language with equality of Peano arithmetic. Its variables are denoted by $x, y, z, w, \ldots$. Expressions, denoted by e, are defined by $e ::= x \mid 0 \mid 1 \mid e + e \mid e \times e$. A quantifier-free formula b is defined by

$$b ::= e = e \mid e < e \mid \neg b \mid b \wedge b \mid b \vee b \mid b \to b.$$

The formula (of the assertion language), denoted by A, B, C, is defined by

$$A ::= e = e \mid e < e \mid \neg b \mid b \wedge b \mid b \vee b \mid b \to b \mid \forall x A \mid \exists x A.$$

Recursive procedures are denoted by R. *WHILE* programs extended to recursive procedures, denoted by P, Q, is defined in [3] by

$$\begin{array}{rcl} P, Q & ::= & x := e \\ & | & \text{if } (b) \text{ then } (P) \text{ else } (P) \\ & | & \text{while } (b) \text{ do } (P) \\ & | & P; P \\ & | & \text{skip} \\ & | & R. \end{array}$$

We assume that procedure R is declared with its body Q.

The basic formula of Hoare's logic is composed with three elements. They are two assertions A and B and a program P. It is expressed in the form

$$\{A\}P\{B\}$$

that is also called a correctness formula or an asserted program. Here A and B are called the precondition and the postcondition of the program P respectively. Whenever A is true before the execution of P and the execution terminates, B is true after the execution.

2.1.2 Semantics

States, denoted by s, are defined in [3] as a function from the set of variables V to the set of natural numbers N. The semantics of the programming language is defined first by $[\![P]\!]^-$ for the programs that do not contain procedures, which is a partial function from States to States.

The semantics of assertions is denoted by $[\![A]\!]_s$ that gives us the truth value of A at the state s.

Definition 2.1.1 *The definition of semantics of programs is given below.*

$$
\begin{array}{lcl}
[\![x := e]\!]^-(s) & = & s[x := [\![e]\!]_s], \\
[\![\mathit{if}\ (b)\ \mathit{then}\ (P_1)\ \mathit{else}\ (P_2)]\!]^-(s) & = & \begin{cases} [\![P_1]\!](s) & \mathit{if}\ [\![b]\!]_s = \mathit{True} \\ [\![P_2]\!](s) & \mathit{otherwise}, \end{cases} \\
[\![\mathit{while}\ (b)\ \mathit{do}\ (P)]\!]^- & = & \begin{cases} s & \mathit{if}\ [\![b]\!]_s = \mathit{False} \\ [\![\mathit{while}\ (b)\ \mathit{do}\ (P)]\!]^-([\![P]\!]^-(s)) & \mathit{otherwise}, \end{cases} \\
[\![P_1; P_2]\!]^-(s) & = & [\![P_2]\!]^-([\![P_1]\!]^-(s)) \\
[\![\mathit{skip}]\!]^-(s) & = & s.
\end{array}
$$

In order to define the semantics of programs which include recursive procedures, Apt provided the approximation semantics of programs. He defined a procedure-less program $P^{(n)}$ by induction on n:

$$
\begin{array}{lcl}
P^{(0)} & = & \Omega, \\
P^{(n+1)} & = & P[Q^{(n)}/R].
\end{array}
$$

He then defined the semantics of programs by

$$
[\![P]\!] \quad = \quad \bigcup_{i=0}^{\infty} [\![P[Q^{(i)}/R]]\!]^-
$$

An asserted program (or a correctness formula) $\{A\}P\{B\}$ is defined to be true if and only if for all states s, s', if $[\![A]\!]_s = \text{True}$ and $[\![P]\!](s) = s'$ then $[\![B]\!]_{s'} = \text{True}$.

2.1.3 Logical System

The logical system H given in [3] consists of the following axioms and inference rules. Here Γ is used as a set of asserted programs. A judgment is defined as $\Gamma \vdash \{A\}P\{B\}$. $\mathit{var}(P)$ is defined as all the variables appeared in the execution of P.

AXIOM 1: ASSIGNMENT AXIOM

$$\frac{}{\Gamma \vdash \{A[x := e]\}x := e\{A\}} \textbf{ (assignment)}$$

RULE 2: COMPOSITION RULE

$$\frac{\Gamma \vdash \{A\}P_1\{C\} \quad \Gamma \vdash \{C\}P_2\{B\}}{\Gamma \vdash \{A\}P_1; P_2\{B\}} \textbf{ (composition)}$$

RULE 3: **if-then-else** RULE

$$\frac{\Gamma \vdash \{A \wedge b\}P_1\{B\} \quad \Gamma \vdash \{A \wedge \neg b\}P_2\{B\}}{\Gamma \vdash \{A\}\text{if}\,(b)\text{ then }(P_1)\text{ else }(P_2)\{B\}} \textbf{ (if-then-else)}$$

RULE 4: **while** RULE

$$\frac{\Gamma \vdash \{A \wedge b\}P\{A\}}{\Gamma \vdash \{A\}\text{while }(b)\text{ do }(P)\{A \wedge \neg b\}} \textbf{ (while)}$$

RULE 5: CONSEQUENCE RULE

$$\frac{\Gamma \vdash \{A_1\}P\{B_1\}}{\Gamma \vdash \{A\}P\{B\}} \textbf{ (consequence)} \quad (A \to A_1, B_1 \to B \text{ true})$$

RULE 8: RECURSION RULE

$$\frac{\Gamma \cup \{A\}R\{B\} \vdash \{A\}Q\{B\}}{\Gamma \vdash \{A\}R\{B\}} \textbf{ (recursion)}$$

AXIOM 9: INVARIANCE AXIOM

$$\frac{}{\Gamma \vdash \{A\}R\{A\}} \textbf{ (invariance)} \quad (\mathrm{FV}(A) \cap \mathit{var}(R) = \emptyset)$$

Rule 10: Substitution Rule I

$$\frac{\Gamma \vdash \{A\}R\{B\}}{\Gamma \vdash \{A[\overrightarrow{z} := \overrightarrow{y}]\}R\{B[\overrightarrow{z} := \overrightarrow{y}]\}}\ \textbf{(substitution I)}\quad (\overrightarrow{y}, \overrightarrow{z} \notin var(R))$$

Rule 11: Substitution Rule II

$$\frac{\Gamma \vdash \{A\}R\{B\}}{\Gamma \vdash \{A[\overrightarrow{z} := \overrightarrow{y}]\}R\{B\}}\ \textbf{(substitution II)}\quad (\overrightarrow{z} \notin var(R) \cup \mathrm{FV}(B))$$

Rule 12: Conjunction Rule

$$\frac{\Gamma \vdash \{A\}R\{B\} \quad \Gamma \vdash \{A'\}R\{C\}}{\Gamma \vdash \{A \wedge A'\}R\{B \wedge C\}}\ \textbf{(conjunction)}$$

An asserted formula $\{A\}x := e\{A[x := e]\}$ may seems fit more for the assignment axiom. But it is not. For an example like $\{x = a \wedge b = a + 1\}x := b\{(x = a \wedge b = a+1)[x := b]\}$, that is $\{x = a \wedge b = a+1\}x := b\{(b = a \wedge b = a+1)\}$, is not obviously true. Rather $\{(x = b)[x := b]\}x := b\{x = b\}$, that is $\{b = b\}x := b\{x = b\}$ is true. The composition rule is similar to the cut rule in concept. When two programs are executed one after another, the postcondition of the former one is the precondition of the later. The precondition of the former one and the postcondition of the later one are preserved for the execution of the composition of those two programs. The if-then-else rule comes from the fact that truth value of b determines the execution of either P_1 or P_2. The rule itself is very natural. The while rule is a bit tricky. Here A is called a loop invariant, which is an assertion that is preserved before and after the execution of P. The truthness of b triggers execution of P and naturally the execution terminates only when b is false.

The consequence rule is not any ordinary rule like others. Here the important fact is that a stronger precondition and a weaker postcondition may replace respectively the precondition and the postcondition of a valid asserted program without affecting its validity. With an example, it may help to understand it better. The asserted program

$\{x = a\}x := x + 1\{x = a + 1\}$ is indeed valid. But from assignment axiom we may get only $\{(x = a+1)[x := x+1]\}x := x+1\{x = a+1\}$, that is $\{x+1 = a+1\}x := x + 1\{x = a + 1\}$. Since $x = a \to x + 1 = a + 1$, with the help of the consequence rule now we finally get $\{x = a\}x := x + 1\{x = a + 1\}$.

Recursion rule states that if the assumption of a valid asserted program for a recursive procedure gives us a valid asserted program for its body, we can say that the asserted program for the procedure is indeed valid. The invariance axiom confirms us that the precondition is preserved in the postcondition if none of its variables is accessed by the execution of a recursive procedure. Substitution rule I allows variable substitution in the assertions in an asserted program if the recursive procedure does not access the substituted and substituting variables. Substitution rules II allows the substitution of only the variables in the precondition if those neither appear in the recursive procedure nor in the postcondition. Conjunction rule allows the preconditions and the postconditions of two asserted programs for the same recursive procedure to be conjoined.

2.1.4 Soundness

In [3], $\vdash_H \{A\}P\{B\}$ denotes the fact that $\{A\}P\{B\}$ is provable in the logical system H, which uses the assumption that all the true assertions are provided (for consequence rule). In his work, the notion of the *truth* of an asserted program is introduced where he chose the standard interpretation of the assertion language with the domain of natural numbers.

Apt called an asserted program *valid* if it is true under all interpretations. He also called a proof rule *sound* if for all interpretations it preserves the truth of asserted programs. Since it is easy to prove that the axioms are valid and the proof rules are sound, it can be said that the logical system is proved to be sound by induction on the length of proofs.

His soundness theorem claims that for every asserted program $\{A\}P\{B\}$ in the logical system H, if $\vdash_H \{A\}P\{B\}$ is provable under the presence of all true assertions then $\{A\}P\{B\}$ is true.

2.1.5 Completeness in the sense of Cook

The strongest postcondition of an assertion and a program and the weakest precondition of a program and an assertion have a key role in defining the completeness in the sense of Cook of such a proof system where general completeness does not hold. Now we will define the strongest postcondition and the weakest precondition.

Definition 2.1.2 *The strongest postcondition of an assertion A and a program P is defined by*

$$SP(A,P) = \{\, s' \mid \exists s(\llbracket A \rrbracket_s \wedge \llbracket P \rrbracket(s) = s') \,\}.$$

The weakest precondition of a program P and an assertion A is defined by

$$WP(P,A) = \{\, s \mid \forall s'(\llbracket P \rrbracket(s) = s' \rightarrow \llbracket A \rrbracket_{s'}) \,\}.$$

Definition 2.1.3 *An assertion language $\mathcal{L}$ is said to be* **expressive relative** *to the set of programs $\mathcal{P}$ if for all assertions $A \in \mathcal{L}$ and programs $P \in \mathcal{P}$, there exists an assertion $S \in \mathcal{L}$ which defines the strongest postcondition $SP(A,P)$.*

Definition 2.1.4 *A proof system $\mathcal{G}$ for a set of programs $\mathcal{P}$ is said to be complete in the sense of Cook if, for all $\mathcal{L}$ such that $\mathcal{L}$ is* **expressive relative** *to $\mathcal{P}$ and for every asserted formula $\{A\}P\{B\}$, if $\{A\}P\{B\}$ is true then $\vdash_{\mathcal{G}} \{A\}P\{B\}$ is provable.*

Apt presented the proof of the completeness of the system in the sense of Cook in [3] using two central lemmas. We will present them and discuss their proof. Assume that $\overrightarrow{x}$ is the sequence of all variables which occur in P and $\overrightarrow{z}$ is a sequence of some new variables and both of their lengths are same. Assume that the assertion language is expressive for the logical system. So, there exists an assertion S that defines the strongest postcondition of $\overrightarrow{x} = \overrightarrow{z}$ and R. The asserted program $\{\overrightarrow{x} = \overrightarrow{z}\}R\{S\}$ is the most general formula for R, since any other true asserted program about R can be derived from $\{\overrightarrow{x} = \overrightarrow{z}\}R\{S\}$. This claim is the contents of the first lemma.

Lemma 2.1.5 (Apt 1981) *if $\{A\}P\{B\}$ is true then $\{\overrightarrow{x} = \overrightarrow{z}\}R\{S\} \vdash \{A\}P\{B\}$ is provable provided that all the true assertions are given.*

Proof. It is proved by induction on P where the most interesting case is $P = R$. Other cases are similar to that of the system H in [3].

Suppose that P is R. Assume $\{A\}R\{B\}$ is true. We have

$$\vdash \{\overrightarrow{x} = \overrightarrow{z}\}R\{S\}.$$

Let A_1 be $A[\overrightarrow{z} := \overrightarrow{u}]$ and B_1 be $B[\overrightarrow{z} := \overrightarrow{u}]$ where $\overrightarrow{u} \notin \mathrm{FV}(B) \cup var(R)$. By invariance axiom,

$$\vdash \{A_1[\overrightarrow{x} := \overrightarrow{z}]\}R\{A_1[\overrightarrow{x} := \overrightarrow{z}]\}$$

is provable. By the conjunction rule,

$$\vdash \{\overrightarrow{x} = \overrightarrow{z} \wedge A_1[\overrightarrow{x} := \overrightarrow{z}]\}R\{S \wedge A_1[\overrightarrow{x} := \overrightarrow{z}]\}$$

is provable since $\mathrm{FV}(A_1[\overrightarrow{x} := \overrightarrow{z}]) \cap var(R) = \emptyset$. We now show that $S \wedge A_1[\overrightarrow{x} := \overrightarrow{z}] \to B_1$.

Assume $[\![S \wedge A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_s = \mathrm{True}$. By definition $[\![S]\!]_s = \mathrm{True}$. By the property of the strongest postcondition, there exists a state s' such that $[\![R_i]\!](s') = s$ and $[\![\overrightarrow{x} = \overrightarrow{z}]\!]_{s'} = \mathrm{True}$.

By invariance axiom,

$$\vdash \{\neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}R\{\neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}$$

is provable. The by conjunction rule

$$\vdash \{\overrightarrow{x} = \overrightarrow{z} \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}R_i\{S \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}.$$

By soundness,

$$\{\overrightarrow{x} = \overrightarrow{z} \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}R_i\{S \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}$$

is true. Now suppose that $\neg[\![A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_{s'} = \mathrm{True}$. Hence $[\![\overrightarrow{x} = \overrightarrow{z} \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_{s'} = \mathrm{True}$. Therefore, $\neg[\![A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_s = \mathrm{True}$. But $[\![A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_s = \mathrm{True}$ by the assumption for s'. It contradicts the assumption and hence $[\![A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_{s'} = \text{True}$.

Since $\overrightarrow{x} = \overrightarrow{z} \wedge A_1[\overrightarrow{x} := \overrightarrow{z}] \to A_1$, we have $[\![A_1]\!]_{s'} = \text{True}$. Then $[\![A]\!]_{s'[\overrightarrow{z} := \overrightarrow{s'(u)}]} = \text{True}$. Then $[\![R]\!](s'[\overrightarrow{z} := \overrightarrow{s'(u)}]) = s[\overrightarrow{z} := \overrightarrow{s(u)}]$ since $\overrightarrow{z}, \overrightarrow{u} \notin var(R)$. Then by definition, $[\![B]\!]_{s[\overrightarrow{z} := \overrightarrow{u}]}$True. Then by definition, $[\![B_1]\!]_s = \text{True}$. Hence $S \wedge A_1[\overrightarrow{x} := \overrightarrow{z}] \to B_1$ is true.

Then by the consequence rule,

$$\vdash \{\overrightarrow{x} = \overrightarrow{z} \wedge A_1[\overrightarrow{x} := \overrightarrow{z}]\}R\{B_1\}$$

is provable. Then by the substitution rule II,

$$\vdash \{\overrightarrow{x} = \overrightarrow{x} \wedge A_1\}R\{B_1\}$$

is provable. Then by the consequence rule,

$$\vdash \{A_1\}R_i\{B_1\}$$

is provable. By the substitution rule I,

$$\vdash \{A_1[\overrightarrow{u} := \overrightarrow{z}]\}R_i\{B_1[\overrightarrow{u} := \overrightarrow{z}]\}.$$

We have $A \to A_1[\overrightarrow{u} := \overrightarrow{z}]$ and $B_1[\overrightarrow{u} := \overrightarrow{z}] \to B$. Then by the consequence rule,

$$\vdash \{A\}P\{B\}$$

provable. □

Lemma 2.1.6 (Apt 1981) *The next lemma in [3] claims that* $\vdash \{\overrightarrow{x} = \overrightarrow{z}\}R\{S\}$ *is provable.*

Proof. By definition of S, $\{\overrightarrow{x} = \overrightarrow{z}\}R\{S\}$ is true and hence $\{\overrightarrow{x} = \overrightarrow{z}\}Q\{S\}$ is true since $[\![R]\!] = [\![Q]\!]$. By the Lemma 2.1.5, $\{\overrightarrow{x} = \overrightarrow{z}\}R\{S\} \vdash \{\overrightarrow{x} = \overrightarrow{z}\}Q\{S\}$ is provable. By the recursion rule, $\vdash \{\overrightarrow{x} = \overrightarrow{z}\}R\{S\}$ is provable. □

The completeness theorem states that if an asserted program $\{A\}P\{B\}$ is true then $\vdash \{A\}P\{B\}$ is provable where all the true assertions are given. It is the central concept of completeness in the sense of Cook.

Theorem 2.1.7 (Apt 1981) *If $\{A\}P\{B\}$ is true then $\vdash \{A\}P\{B\}$ is provable.*

Proof. Assume $\{A\}P\{B\}$ is true. By Lemma 2.1.5, $\{\vec{x} = \vec{z}\}R\{S\} \vdash \{A\}P\{B\}$ is provable. By Lemma 2.1.6, $\vdash \{\vec{x} = \vec{z}\}R\{S\}$ is provable. Then $\vdash \{A\}P\{B\}$ is provable. □

2.2 Separation Logic

In system programming, use of shared mutable data structures is widespread. For three decades, approaches to reasoning about this technique has been studied. Most of them either have extremely complexity or limited applicability. Until the work of Reynolds in 2002 [18], an extension to pointer programs was missing. Reynolds introduced separation logic, which is an extension of Hoare's logic that permits reasoning about pointer programs that have the ability to use shared mutable data structure. He extended the simple *WHILE* programs with commands for allocating, deallocating, accessing and modifying shared resources. He also extended the assertions by incorporating separating conjunction and separating implication that resembles multiplicative conjunction and multiplicative implication in the logic of bunched implication by O'Hearn and Pym [20]. In his work, he also extended Hoare's logic to pointer programs with several sets of logical rules. Although Reynolds provided the logical system and mentioned that it is sound, he did not provide the proof. The detail technical description of separation logic is given in Chapter 4.

Tatsuta et al. gave the detailed proof of completeness in [19]. In his work he has taken all the axioms and rules from basic Hoare's logic and only the backward reasoning axioms from the rules proposed by Reynolds and proved that his system is complete in the sense of Cook. On the way of proving completeness, he also proved the expressiveness of the separation logic for pointer programs.

The work of O'Hearn gives us local reasoning of Programs [21] using frame rule. It is important to simplify verification since it gives an information hiding mechanism. Yang investigated the "adaptation completeness" (completeness of atomic programs) using the frame rule for programs with procedures, which indicates that all properties can be inferred with the rule [23].

This dissertation is based on [3, 19], that intends to extend Hoare's logic and separation logic to mutual recursive procedures, and discuss admissibility of frame rules in it.

3

New Complete System of Hoare's Logic with Recursive Procedures

We introduce a complete system of Hoare's logic with recursive procedures. Apt gave a system for the same purpose and showed its completeness in [3]. Our system is obtained from Apt's system by replacing the INVARIANCE AXIOM, the SUBSTITUTION RULE I, the SUBSTITUTION RULE II, and the CONJUNCTION RULE by the rules (Inv-Conj) and (Exists). Apt suggested without proofs that one could replace them by his SUBSTITUTION RULE I, (Inv-Conj), and (Exists) to get another complete system. We prove that the substitution rule I can actually be derived in our system. We also give a detailed proof of the completeness of our system.

3.1 Language

Our assertion is a formula A of Peano arithmetic. We define the language $\mathcal{L}$ as follows.

Definition 3.1.1 *Formulas A are defined by*

$$A ::= e = e \mid e < e \mid \neg A \mid A \wedge A \mid A \vee A \mid A \to A \mid \forall x A \mid \exists x A$$

We will sometimes call a formula an assertion.

We define $FV(A)$ as the set of free variables in A. We define $FV(e)$ similarly.

Our program is a while-program with parameterless recursive procedures.

Definition 3.1.2 *Programs, denoted by P,Q, are defined by*

$$\begin{array}{rcl} P, Q & ::= & x := e \\ & \mid & \textit{if}\ (b)\ \textit{then}\ (P)\ \textit{else}\ (P) \\ & \mid & \textit{while}\ (b)\ \textit{do}\ (P) \\ & \mid & P; P \\ & \mid & \textit{skip} \\ & \mid & R_i. \end{array}$$

b is a formula without the quantifiers. R_i is a parameter-less procedure name having Q_i as its definition body. We define the language $\mathcal{L}^-$ as $\mathcal{L}$ excluding the construct R.

An asserted program is defined by $\{A\}P\{B\}$, which means the partial correctness.

3.2 Semantics

Definition 3.2.1 *We define the semantics of our programming language. For a program P, its meaning $[\![P]\!]$ is defined as a partial function from States to States. We will define*

$[\![P]\!](r_1)$ as the resulting state after termination of the execution of P with the initial state r_1. If the execution of P with the initial state r_1 does not terminate, we leave $[\![P]\!](r_1)$ undefined. In order to define $[\![P]\!]$, we would like to define $[\![P]\!]^-$ for all P in the language L^-. We define $[\![P]\!]^-$ by induction on P in $\mathcal{L}^-$ as follows:

$$
\begin{array}{lcl}
[\![x := e]\!]^-(s) & = & s[x := [\![e]\!]_s], \\
[\![if\,(b)\; then\; (P_1)\; else\; (P_2)]\!]^-(s) & = & \begin{cases} [\![P_1]\!](s) & if\; [\![b]\!]_s = True \\ [\![P_2]\!](s) & otherwise, \end{cases} \\
[\![while\;(b)\;do\;(P)]\!]^- & = & \begin{cases} s & if\; [\![b]\!]_s = False \\ [\![while\;(b)\;do\;(P)]\!]^-([\![P]\!]^-(s)) & otherwise, \end{cases} \\
[\![P_1; P_2]\!]^-(s) & = & [\![P_2]\!]^-([\![P_1]\!]^-(s)) \\
[\![skip]\!]^-(s) & = & s.
\end{array}
$$

Definition 3.2.2 *For an asserted program $\{A\}P\{B\}$, the meaning of $\{A\}P\{B\}$ is defined as True or False. $\{A\}P\{B\}$ is defined to be True if the following holds.*

For all s and s', if $[\![A]\!]_s = True$ and $[\![P]\!](s) = s'$, then $[\![B]\!]_{s'} = True$.

Definition 3.2.3 *The semantics of P in $\mathcal{L}$ is defined by*

$$
[\![P]\!](s) = \begin{cases} s' & if\; \{[\![P^{(i)}]\!]^-(s) | i \geq 0\} = \{s'\} \\ undefined & if\; \{[\![P^{(i)}]\!]^-(s) | i \geq 0\} = \emptyset \end{cases}
$$

3.3 Logical System

This section defines the logical system.

We will write $A[x := e]$ for the formula obtained from A by replacing x by e.

Definition 3.3.1 *Our logical system consists of the following inference rules. As mentioned in previous section, we will use Γ for a set of asserted programs. A judgment is defined as $\Gamma \vdash \{A\}P\{B\}$.*

Skip

$$\overline{\Gamma \vdash \{A\} skip \{A\}}$$

Identity

$$\overline{\Gamma, \{A\}P\{B\} \vdash \{A\}P\{B\}}$$

Assignment

$$\overline{\Gamma \vdash \{A[x := e]\} x := e \{A\}}$$

If

$$\frac{\Gamma \vdash \{A \wedge b\}P_1\{B\} \quad \Gamma \vdash \{A \wedge \neg b\}P_2\{B\}}{\Gamma \vdash \{A\} if\,(b)\ then\ (P_1)\ else\ (P_2)\{B\}}$$

While

$$\frac{\Gamma \vdash \{A \wedge b\}P\{A\}}{\Gamma \vdash \{A\} while\,(b)\ do\ (P)\{A \wedge \neg b\}}$$

Composition

$$\frac{\Gamma \vdash \{A\}P_1\{C\} \quad \Gamma \vdash \{C\}P_2\{B\}}{\Gamma \vdash \{A\}P_1; P_2\{B\}}$$

Conseq

$$\frac{\Gamma \vdash \{A_1\}P\{B_1\}}{\Gamma \vdash \{A\}P\{B\}}\ (A \to A_1, B_1 \to B)$$

Recursion

$$\frac{\begin{array}{c}\Gamma \cup \{\{A_i\}R_i\{B_i\} | i = 1, \ldots, n_{proc}\} \vdash \{A_1\}Q_1\{B_1\} \\ \vdots \\ \Gamma \cup \{\{A_i\}R_i\{B_i\} | i = 1, \ldots, n_{proc}\} \vdash \{A_{n_{proc}}\}Q_{n_{proc}}\{B_{n_{proc}}\}\end{array}}{\Gamma \vdash \{A_j\}R_j\{B_j\}}\ 1 \leq j \leq n_{proc}$$

Inv-Conj

$$\frac{\Gamma \vdash \{A\}P\{C\}}{\Gamma \vdash \{A \wedge B\}P\{C \wedge B\}} \ (FV(B) \cap Mod(P) = \emptyset)$$

Exists

$$\frac{\Gamma \vdash \{A\}P\{B\}}{\Gamma \vdash \{\exists x.A\}P\{B\}} \ (x \notin FV(B) \cup EFV(P))$$

We say $\{A\}P\{B\}$ is provable and we write $\vdash \{A\}P\{B\}$, when $\vdash \{A\}P\{B\}$ can be derived by these inference rules.

The rule (Exists) is analogous to the rule existential introduction of propositional calculus.

3.4 Completeness

Lemma 3.4.1 *If $\{A\}P_1\{B\}$ is true and $[\![P_1]\!] = [\![P_2]\!]$ then $\{A\}P_2\{B\}$ is true.*

Proof. By definition. □

Definition 3.4.2 *X is called the strongest postcondition of P and A if and only if the following holds.*

(1) For all s, s', if $[\![A]\!]_s = True$ and $[\![P]\!](s) = s'$ then $s' \in X$.

(2) For all Y, if $\forall s, s'([\![A]\!]_s = True \wedge [\![P]\!](s) = s' \to s' \in Y)$ then $X \subseteq Y$.

Definition 3.4.3 *$S_{A,P}(\overrightarrow{x})$ is defined as the strongest postcondition for A and P.*

$S_{A,P}(\overrightarrow{x})$ gives the strongest assertion S such that $\{A\}P\{S\}$ is true.

Lemma 3.4.4 *If $\vdash \{A\}P\{B\}$ then $\vdash \{A[\overrightarrow{x} := \overrightarrow{z}]\}P\{B[\overrightarrow{x} := \overrightarrow{z}]\}$ where $\overrightarrow{z}, \overrightarrow{x} \notin EFV(P)$.*

Proof. Assume $\vdash \{A\}P\{B\}$ and $\overrightarrow{z} \notin \mathrm{EFV}(P)$. Then by (Inv-Conj),

$$\vdash \{A \wedge \overrightarrow{x} = \overrightarrow{z}\}P\{B \wedge \overrightarrow{x} = \overrightarrow{z}\}.$$

We have $B \wedge \overrightarrow{x} = \overrightarrow{z} \to B[\overrightarrow{x} := \overrightarrow{z}]$. Then by (Conseq),

$$\vdash \{A \wedge \overrightarrow{x} = \overrightarrow{z}\}P\{B[\overrightarrow{x} := \overrightarrow{z}]\}.$$

Then by (Exists),

$$\vdash \{\exists \overrightarrow{z}(A \wedge \overrightarrow{x} = \overrightarrow{z})\}P\{B[\overrightarrow{x} := \overrightarrow{z}]\}.$$

We have $A[\overrightarrow{x} := \overrightarrow{z}] \to \exists \overrightarrow{z}(A \wedge \overrightarrow{x} = \overrightarrow{z})$. Then by (Conseq),

$$\vdash \{A[\overrightarrow{x} := \overrightarrow{z}]\}P\{B[\overrightarrow{x} := \overrightarrow{z}]\}.$$

□

Lemma 3.4.5 *If $\{A\}P\{B\}$ is true and $\overrightarrow{z} \notin EFV(P)$ then $\Gamma \vdash \{A\}P\{B\}$ where $\Gamma = \{\{\overrightarrow{x} = \overrightarrow{z}\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x})\} | i = 1,\dots,n\}$, $\overrightarrow{x} = x_1,\dots,x_m$ and $\{x_j | j = 1,\dots,m\} = EFV(P)$.*

Proof. We will prove it by induction on P. We will consider the cases of P.

If P is other than R_i, the proof of these cases are similar to those of completeness proof of H given in [3].

Case P is R_i.

Assume $\{A\}R_i\{B\}$ is true and $\vec{z} \notin \mathrm{EFV}(R_i)$. We have

$$\Gamma \vdash \{\overrightarrow{x} = \overrightarrow{z}\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x})\}.$$

Let A_1 be $A[\overrightarrow{z} := \overrightarrow{u}]$ and B_1 be $B[\overrightarrow{z} := \overrightarrow{u}]$ where $\overrightarrow{u} \notin \mathrm{FV}(B) \cup \mathrm{EFV}(R_i)$. By the rule (Inv-Conj),

$$\Gamma \vdash \{\overrightarrow{x} = \overrightarrow{z} \wedge A_1[\overrightarrow{x} := \overrightarrow{z}]\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x}) \wedge A_1[\overrightarrow{x} := \overrightarrow{z}]\}$$

since $\mathrm{FV}(A_1[\overrightarrow{x} := \overrightarrow{z}]) \cap \mathrm{Mod}(R_i) = \emptyset$.

We now show that $S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x}) \wedge A_1[\overrightarrow{x} := \overrightarrow{z}] \to B_1$.

Assume $[\![S_{\overrightarrow{x}=\overrightarrow{z},R}(\overrightarrow{x}) \wedge A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_s = \mathrm{True}$. By definition $[\![S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x})]\!]_s = \mathrm{True}$. By the property of the strongest postcondition, there exists a state s' such that $[\![R_i]\!](s') = s$ and $[\![\overrightarrow{x} = \overrightarrow{z}]\!]_{s'} = \mathrm{True}$.

Now suppose that $\neg[\![A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_{s'} = \mathrm{True}$. By (Inv-Conj),

$$\Gamma \vdash \{\overrightarrow{x} = \overrightarrow{z} \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x}) \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}.$$

Since Γ is true by definition, by soundness, $\{\overrightarrow{x} = \overrightarrow{z} \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x}) \wedge \neg A_1[\overrightarrow{x} := \overrightarrow{z}]\}$ is true. Then by definition $\neg[\![A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_s = \mathrm{True}$. But $[\![A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_s = \mathrm{True}$. It contradicts the assumption and hence $[\![A_1[\overrightarrow{x} := \overrightarrow{z}]]\!]_{s'} = \mathrm{True}$.

Since $\overrightarrow{x} = \overrightarrow{z} \wedge A_1[\overrightarrow{x} := \overrightarrow{z}] \to A_1$, we have $[\![A_1]\!]_{s'} = \mathrm{True}$. Then $[\![A]\!]_{s'[\overrightarrow{z}:=\overrightarrow{s'(u)}]} = \mathrm{True}$. Then $[\![R_i]\!](s'[\overrightarrow{z} := \overrightarrow{s'(u)}]) = s[\overrightarrow{z} := \overrightarrow{s(u)}]$ since $\overrightarrow{z}, \overrightarrow{u} \notin \mathrm{EFV}(R_i)$. Then by definition, $[\![B]\!]_{s[\overrightarrow{z}:=\overrightarrow{u}]}$True. Then by definition, $[\![B_1]\!]_s = \mathrm{True}$. Hence $S_{\overrightarrow{x}=\overrightarrow{z},R}(\overrightarrow{x}) \wedge A_1[\overrightarrow{x} := \overrightarrow{z}] \to B_1$ is true. Then by the rule (Conseq),

$$\Gamma \vdash \{\overrightarrow{x} = \overrightarrow{z} \wedge A_1[\overrightarrow{x} := \overrightarrow{z}]\}R_i\{B_1\}.$$

Then by the rule (Exists),

$$\Gamma \vdash \{\exists \overrightarrow{z}(\overrightarrow{x} = \overrightarrow{z} \wedge A_1[\overrightarrow{x} := \overrightarrow{z}])\}R_i\{B_1\}.$$

We have $A_1 \to \exists \overrightarrow{z}(\overrightarrow{x} = \overrightarrow{z} \wedge A_1[\overrightarrow{x} := \overrightarrow{z}])$. Then by the rule (Conseq),

$$\Gamma \vdash \{A_1\}R_i\{B_1\}.$$

By Lemma 3.4.4,

$$\Gamma \vdash \{A_1[\overrightarrow{u} := \overrightarrow{z}]\}R_i\{B_1[\overrightarrow{u} := \overrightarrow{z}]\}.$$

We have $A \to A_1[\overrightarrow{u} := \overrightarrow{z}]$ and $B_1[\overrightarrow{u} := \overrightarrow{z}] \to B$. Then by (conseq),

$$\Gamma \vdash \{A\}P\{B\},$$

which was to be proved. $\square$

Next lemma shows that the hypothesis $\{\overrightarrow{x} = \overrightarrow{z}(\overrightarrow{x})\}R\{S_{\overrightarrow{x}=\overrightarrow{z},R}(\overrightarrow{x})\}$ used in lemma 5.3.7 is provable in the our system.

Lemma 3.4.6 *$\vdash \{\overrightarrow{x} = \overrightarrow{z}\}R_j\{S_{\overrightarrow{x}=\overrightarrow{z},R_j}(\overrightarrow{x})\}$ for $j = 1, \ldots, n$ where $\overrightarrow{x} = x_1, \ldots, x_m$, $\{x_j | j = 1, \ldots, m\} = \bigcup_{i=1}^n EFV(R_i)$ and $\overrightarrow{z} \notin \bigcup_{i=1}^n EFV(R_i)$.*

Proof. Assume $\overrightarrow{z} \notin \bigcup_{i=1}^n \mathrm{EFV}(R_i)$ and $\overrightarrow{x} = x_1, \ldots, x_m$ where $\{x_j | j = 1, \ldots, m\} = \bigcup_{i=1}^n \mathrm{EFV}(R_i)$.

Fix j. Assume $[\![\overrightarrow{x} = \overrightarrow{z}]\!]_s = \mathrm{True}$. Assume $[\![Q_j]\!](s) = r$ where Q_j is the body of R_j. Then by Lemma 3.4.1, $[\![R_j]\!](s) = r$. By definition, $[\![S_{\overrightarrow{x}=\overrightarrow{z},R_j}(\overrightarrow{x})]\!]_r = \mathrm{True}$. Then by definition, $\{\overrightarrow{x} = \overrightarrow{z}\}Q_j\{S_{\overrightarrow{x}=\overrightarrow{z},R_j}(\overrightarrow{x})\}$ is true. By Lemma 3.4.5, $\{\{\overrightarrow{x} = \overrightarrow{z}\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x})\} | i = 1, \ldots, n\} \vdash \{\overrightarrow{x} = \overrightarrow{z}\}Q_j\{S_{\overrightarrow{x}=\overrightarrow{z},R_j}(\overrightarrow{x})\}$. Hence, $\{\{\overrightarrow{x} = \overrightarrow{z}\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x})\} | i = 1, \ldots, n\} \vdash \{\overrightarrow{x} = \overrightarrow{z}\}Q_j\{S_{\overrightarrow{x}=\overrightarrow{z},R_j}(\overrightarrow{x})\}$ for all $j = 1, \ldots, n$. Then by the rule (Recursion), $\vdash \{\overrightarrow{x} = \overrightarrow{z}\}R_j\{S_{\overrightarrow{x}=\overrightarrow{z},R_j}(\overrightarrow{x})\}$. $\square$

The following theorem is the key theorem of this paper. It says that our system is complete.

Theorem 3.4.7 *If $\{A\}P\{B\}$ is true then $\vdash \{A\}P\{B\}$ is provable.*

Proof. Assume $\{A\}P\{B\}$ is true. Let $\overrightarrow{z}$ be such that $\overrightarrow{z} \notin \bigcup_{i=1}^n \mathrm{EFV}(R_i) \cup \mathrm{EFV}(P)$ and $\overrightarrow{x} = x_1, \ldots, x_m$ where $\{x_j | j = 1, \ldots, m\} = \bigcup_{i=1}^n \mathrm{EFV}(R_i) \cup \mathrm{EFV}(P)$. Then by Lemma 3.4.5, $\{\{\overrightarrow{x} = \overrightarrow{z}\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x})\} | i = 1, \ldots, n\} \vdash \{A\}P\{B\}$. By Lemma 3.4.6, $\vdash \{\overrightarrow{x} = \overrightarrow{z}\}R_i\{S_{\overrightarrow{x}=\overrightarrow{z},R_i}(\overrightarrow{x})\}$ for $i = 1, \ldots, n$. Then we have $\vdash \{A\}P\{B\}$. $\square$

Apt's system cannot be extended to separation logic, because his invariance axiom is inconsistent with separation logic. On the other hand, we can extend our system to a verification system with separation logic and recursive procedures in a straightforward way.

4

Separation Logic for Recursive Procedures

4.1 Language

This section defines our programming language and our assertion language. Our programming language inherits from the pointer programs in Reynolds' paper [18]. Our assertion language is also the same as in [18], which is based on Peano arithmetic.

4.1.1 Base Language

We first define our base language, which will be used later for both a part of our programming language and a part of our assertion language. It is essentially a first-order language for Peano arithmetic. We call its formula a *pure* formula. We will use i, j, k, l, m, n for natural numbers. Our base language is defined as follows. We have variables $x, y, z, w, \ldots$ and constants $0, 1$, null, denoted by c. The symbol null means the null pointer. We have function symbols $+, \times$ and we do not have any predicate constants. Our predicate symbols are $=$ and $<$. Terms and expressions, denoted by e, are defined by $e ::= \ x \mid c \mid e + e \mid e \times e$. Terms mean natural numbers or pointers. Our *pure* formulas, denoted by A, are defined by

$$A ::= \ e = e \mid e < e \mid \neg A \mid A \wedge A \mid A \vee A \mid A \to A \mid \forall x A \mid \exists x A.$$

The formula constructions mean usual logical connectives. We will sometimes write the number n to denote the term $1 + (1 + (1 + \ldots (1 + 0)))$ \quad (n times of $1+$).

4.1.2 Programming Language

Next we define our programming language, which is an extension of while programs to pointers and procedures. Its expressions are terms of the base language. Its boolean expressions, denoted by b, are quantifier-free *pure* formulas and defined by $b ::= \ e = e \mid e < e \mid \neg b \mid b \wedge b \mid b \vee b \mid b \to b$. Boolean expressions are used as conditions in a program.

We assume procedure names $R_1, \ldots, R_{n_{proc}}$ for some n_{proc}. We will write R for these procedure names.

Definition 4.1.1 *Programs, denoted by P,Q, are defined by*

$$
\begin{array}{rcll}
P & ::= & x := e & (assignment) \\
& | & \mathit{if}\,(b)\ \mathit{then}\ (P)\ \mathit{else}\ (P) & (conditional) \\
& | & \mathit{while}\ (b)\ \mathit{do}\ (P) & (iteration) \\
& | & P; P & (composition) \\
& | & \mathit{skip} & (no\ operation) \\
& | & x := \mathit{cons}(e, e) & (allocation) \\
& | & x := [e] & (lookup) \\
& | & [e] := e & (mutation) \\
& | & \mathit{dispose}(e) & (deallocation) \\
& | & R & (mutual\ recursive\ procedure\ name)
\end{array}
$$

R means a procedure name without parameters.

We write $\mathcal{L}$ for the set of programs. We write $\mathcal{L}^-$ for the set of programs that do not contain procedure names.

The statement $x := \mathrm{cons}(e_1, e_2)$ allocates two new consecutive memory cells, puts the values of e_1 and e_2 in the respective cells, and assigns the first address to x. The statement $x := [e]$ looks up the content of the memory cell at the address e and assigns it to x. The statement $[e_1] := e_2$ changes the content of the memory cell at the address e_1 by e_2. The statement $\mathrm{dispose}(e)$ deallocates the memory cell at the address e.

The programs $x := e$, skip, $x := \mathrm{cons}(e_1, e_2)$, $x := [e]$, $[e_1] := e_2$ and $\mathrm{dispose}(e)$ are called *atomic* programs.

We call Procedure $R(Q)$ a procedure declaration where R is a procedure name and Q is a program. The program Q is said to be the body of R. This means that we define the procedure name R with its procedure body Q.

We assume the procedure declarations

$$\{\text{Procedure } R_1(Q_1), \ldots, \text{Procedure } R_{n_{proc}}(Q_{n_{proc}})\}$$

that gives procedure definitions to all procedure names in the rest of the paper. We allow mutual recursive procedure calls.

4.1.3 Assertion Language and Asserted Programs

Our assertion language is a first-order language extended by the separating conjunction $*$ and the separating implication $-\!*$ as well as emp and $\mapsto$. Its variables, constants, function symbols, and terms are the same as those of the base language. We have predicate symbols $=$, $<$ and $\mapsto$ and a predicate constant emp. Our assertion language is defined as follows.

Definition 4.1.2 *Formulas A are defined by*

$$A ::= emp \mid e = e \mid e < e \mid e \mapsto e \mid \neg A \mid A \wedge A \mid A \vee A \mid A \to A \mid \forall x A \mid \exists x A \mid A * A \mid A -\!* A$$

We will sometimes call a formula an assertion.

We define $FV(A)$ as the set of free variables in A. We define $FV(e)$ similarly.

The symbol emp means the current heap is empty. The formula $e_1 \mapsto e_2$ means the current heap has only one cell at the address e_1 and its content is e_2. The formula $A * B$ means the current heap can be split into some two disjoint heaps such that the formula A holds at one heap and the formula B holds at the other heap. The formula $A -\!* B$ means that for any heap disjoint from the current heap such that the formula A holds at the heap, the formula B holds at the new heap obtained from the current heap and the heap by combining them.

We use vector notation to denote a sequence. For example, $\overrightarrow{e}$ denotes the sequence $e_1, \ldots, e_n$ of expressions.

Definition 4.1.3 *The expression* $\{A\}P\{B\}$ *is called an asserted program, where* A, B *are formulas and* P *is a program.*

This means the program P with its precondition A and its postcondition B.

4.1.4 Unfolding of Procedures

We define the set of procedure names which are visible in a program. It will be necessary later in defining the dependency relation between two procedures.

Definition 4.1.4 *The set* $PN(P)$ *of procedure names in* P *is defined as follows.*

$$
\begin{array}{lcll}
PN(P) & = & \emptyset & \textit{if } P \textit{ is atomic}, \\
PN(\textit{if }(b)\textit{ then }(P_1)\textit{ else }(P_2)) & = & PN(P_1) \cup PN(P_2), & \\
PN(P_1; P_2) & = & PN(P_1) \cup PN(P_2), & \\
PN(\textit{while }(b)\textit{ do }(P)) & = & PN(P), & \\
PN(R_i) & = & \{R_i\}. &
\end{array}
$$

We define the dependency relation between two procedures. When a procedure name appears in the body of another procedure, we say the latter procedure depends on the former procedure at level 1. When one procedure depends on another and the latter one again depends on the third one, we say the first one also depends on the third one. In this case, the level of the third dependency is determined by summing up the levels of first and second dependencies mentioned above.

Definition 4.1.5 *We define the relation* $R_i \overset{k}{\leadsto} R_j$ *as follows:*

$$
\begin{array}{l}
R_i \overset{0}{\leadsto} R_i, \\
R_i \overset{1}{\leadsto} R_j \textit{ if } PN(Q_i) \ni R_j, \\
R_i \overset{k}{\leadsto} R_j \textit{ if } R_i = R'_0 \overset{1}{\leadsto} R'_1 \overset{1}{\leadsto} \ldots \overset{1}{\leadsto} R'_k = R_j \textit{ for some } R'_0, \ldots, R'_k.
\end{array}
$$

Procedures dependency $PD(R_i, k)$ *of a procedure name* R_i *up to level* k *is defined by* $PD(R_i, k) = \{ R_j \mid R_i \overset{l}{\leadsto} R_j, l \leq k \}$.

This relation will be used to define EFV(P) and Mod(P) as well as the semantics of P.

Note that (1) $\mathrm{PD}(R_i, k) \subseteq \mathrm{PD}(R_i, k+1)$ for all k and (2) $\mathrm{PD}(R_i, k) \subseteq \{ R_i \mid i = 1, \ldots, n_{proc} \}$ where n_{proc} is the number of procedures in the declaration.

The following first lemma will show that once procedures dependencies of a procedure up to two consecutive levels are the same, it is the same up to any higher level too. The second claim states that $n_{proc} - 1$ is sufficient for the largest level.

Lemma 4.1.6 *(1) If $PD(R_i, k) = PD(R_i, k+1)$ then $PD(R_i, k) = PD(R_i, k+l)$ for all $k, l \in N$.*

(2) $PD(R_i, k) \subseteq PD(R_i, n_{proc} - 1)$ for all k.

Proof. (1) It is proved by induction on l.

Case 1. l be 0.

Its proof is immediate.

Case 2. l be $l' + 1$.

Assume $\mathrm{PD}(R_i, k) = \mathrm{PD}(R_i, k+1)$. We can show that if $R_i' \in \mathrm{PD}(R_i, k)$ then $R_i' \in \mathrm{PD}(R_i, k + l' + 1)$. Now we will show that if $R_i' \in \mathrm{PD}(R_i, k + l' + 1)$ then $R_i' \in \mathrm{PD}(R_i, k)$. Assume $R_i' \in \mathrm{PD}(R_i, k + l' + 1)$. Then we have R_j such that $R_j \in \mathrm{PD}(R_i, k + l')$ and $R_j \overset{1}{\leadsto} R_i'$. By induction hypothesis, $\mathrm{PD}(R_i, k) = \mathrm{PD}(R_i, k + l')$. Then $R_j \in \mathrm{PD}(R_i, k)$. Then by definition, $R_i' \in \mathrm{PD}(R_i, k+1)$. Then $R_i' \in \mathrm{PD}(R_i, k)$ by the assumption. Therefore, $\mathrm{PD}(R_i, k) = \mathrm{PD}(R_i, k + l' + 1)$.

(2) We will show that $\mathrm{PD}(R_i, m) = \mathrm{PD}(R_i, m+1)$ for some $m < n_{proc}$ by contradiction. Assume for all $m < n_{proc}$, $\mathrm{PD}(R_i, m) \neq \mathrm{PD}(R_i, m+1)$. Then $\mathrm{PD}(R_i, m) \subsetneqq \mathrm{PD}(R_i, m+1)$. Then we have $\mid \mathrm{PD}(R_i, m) \mid \geq m + 1$ and hence $\mid \mathrm{PD}(R_i, n_{proc}) \mid \geq n_{proc} + 1$. But $\mathrm{PD}(R_i, n_{proc}) \subseteq \{ R_i \mid i = 1, \ldots, n_{proc} \}$ and then $\mid \mathrm{PD}(R_i, n_{proc}) \mid \leq n_{proc}$. It is a contradiction. Therefore, $\mathrm{PD}(R_i, m) = \mathrm{PD}(R_i, m+1)$ for some $m < n_{proc}$. By (1), we have now $\mathrm{PD}(R_i, n_{proc} - 1) = \mathrm{PD}(R_i, n_{proc} - 1 + l)$ for all l.

Therefore, $\mathrm{PD}(R_i, k) \subseteq \mathrm{PD}(R_i, n_{proc} - 1)$ for all k. $\square$

We need to define some closed program that never terminates in order to define unfolding of a program for a specific number of times. First we will define Ω.

Definition 4.1.7 *We define Ω as*

$$\textit{while}\ (0 = 0)\ \textit{do}\ (\textit{skip})$$

Substitution of a program for a procedure name is defined below.

Definition 4.1.8 *Let $\overrightarrow{P}' = P'_1, \ldots, P'_{n_{proc}}$ where P'_i is a program. $P[\overrightarrow{P}']$ is defined by induction on P as follows:*

$$\begin{array}{lcl}
P[\overrightarrow{P}'] & = & P \qquad \textit{if } P \textit{ is atomic}, \\
(\textit{if}\ (b)\ \textit{then}\ (P_1)\ \textit{else}\ (P_2))[\overrightarrow{P}'] & = & (\textit{if}\ (b)\ \textit{then}\ (P_1[\overrightarrow{P}'])\ \textit{else}\ (P_2[\overrightarrow{P}'])), \\
(\textit{while}\ (b)\ \textit{do}\ (P))[\overrightarrow{P}'] & = & (\textit{while}\ (b)\ \textit{do}\ (P[\overrightarrow{P}'])), \\
(P_1; P_2)[\overrightarrow{P}'] & = & (P_1[\overrightarrow{P}']; P_2[\overrightarrow{P}']), \\
(R_i)[\overrightarrow{P}'] & = & P'_i.
\end{array}$$

$P[\overrightarrow{P}']$ is a program obtained from P by replacing the procedure names $R_1, \ldots, R_{n_{proc}}$ by $P'_1, \ldots, P'_{n_{proc}}$ respectively.

Unfolding transforms a program in language $\mathcal{L}$ into a program in language $\mathcal{L}^-$. Discussions on the programs in language $\mathcal{L}$ can be reduced to those in $\mathcal{L}^-$, which are either easy or already shown elsewhere. $P^{(k)}$ denotes P where each procedure name is unfolded only k times. $P^{(0)}$ just replaces a procedure name by Ω, since a procedure name is not unfolded any more, which means the procedure name is supposed to be not executed. Here we present the unfolding of a program.

Definition 4.1.9 *Let $\Omega_i = \Omega$ for $1 \le i \le n_{proc}$. We define $P^{(k)}$ for $k \ge 0$ as follows:*

$$\begin{array}{lcl}
P^{(0)} & = & P[\Omega_1, \ldots, \Omega_{n_{proc}}], \\
P^{(k+1)} & = & P[Q_1^{(k)}, \ldots, Q_{n_{proc}}^{(k)}].
\end{array}$$

Sometimes we will call $P^{(k)}$ as the k-times unfolding of the program P.

We present some basic properties of unfolded programs.

Proposition 4.1.10 *(1)* $P^{(k)} = P[\overrightarrow{R^{(k)}}]$.

(2) $R_i^{(0)} = \Omega$.

(3) $R_i^{(k+1)} = Q_i[\overrightarrow{R^{(k)}}]$.

Proof. (1) By case analysis of k.

Case 1. $k = 0$.

$P^{(0)} = P[\overrightarrow{\Omega}]$ by definition. By (2), $P[\overrightarrow{\Omega}] = P[\overrightarrow{R^{(0)}}]$. Therefore, $P^{(0)} = P[\overrightarrow{R^{(0)}}]$.

Case 2. $k = k' + 1$.

$P^{(k'+1)} = P[\overrightarrow{Q^{(k')}}]$. Since $R_i^{(k'+1)} = R_i[\overrightarrow{Q^{(k')}}] = Q_i^{(k')}$, $P[\overrightarrow{Q^{(k')}}] = P[\overrightarrow{R^{(k'+1)}}]$. Therefore, $P^{(k'+1)} = P[\overrightarrow{R^{(k'+1)}}]$.

(2) By definition we have $R_i^{(0)} = R_i[\overrightarrow{\Omega}] = \Omega$.

(3) By definition we have $R_i^{(k+1)} = R_i[\overrightarrow{Q^{(k)}}] = Q_i^{(k)}$. By (1), $Q_i^{(k)} = Q_i[\overrightarrow{R^{(k)}}]$. Therefore, $R_i^{(k+1)} = Q_i[\overrightarrow{R^{(k)}}]$. □

The next two definitions will define the set of the free variables $(\mathrm{FV}(P))$ and the set of the variables that can be modified $(\mathrm{Mod}_1(P))$. Generally speaking, the left variable of the symbol := in a program is a modifiable variable. First we will define above mentioned two sets for a program in its syntactic structure. Next, it will be used to define the set of free variables (extended free variables, EFV) and the set of modifiable variables $(\mathrm{Mod}(P))$ that may appear in the execution of the program. Since 'a free variable' has an ordinary meaning without procedure calls, we will use 'an extended free variable' for that with procedure calls.

Definition 4.1.11 *We define $FV(P)$ for P in $\mathcal{L}^-$ and $EFV(P)$ for P in $\mathcal{L}$ as follows:*

$$
\begin{array}{lcl}
FV(x := e) & = & \{x\} \cup FV(e), \\
FV(\mathit{if}\ (b)\ \mathit{then}\ (P_1)\ \mathit{else}\ \ (P_2)) & = & FV(b) \cup FV(P_1) \cup FV(P_2), \\
FV(\mathit{while}\ (b)\ \mathit{do}\ (P)) & = & FV(b) \cup FV(P), \\
FV(P_1; P_2) & = & FV(P_1) \cup FV(P_2), \\
FV(\mathit{skip}) & = & \emptyset, \\
FV(x := \mathit{cons}(e_1, e_2)) & = & \{x\} \cup FV(e_1) \cup FV(e_2), \\
FV(x := [e]) & = & \{x\} \cup FV(e), \\
FV([e_1] := e_2) & = & FV(e_1) \cup FV(e_2), \\
FV(\mathit{dispose}(e)) & = & FV(e), \\
EFV(P) & = & FV(P^{(n_{proc})}).
\end{array}
$$

The expression $\mathrm{FV}(P)$ is the set of variables that occur in P. $\mathrm{EFV}(P)$ is the set of variables that may be used in the execution of P.

The expression $\mathrm{FV}(O_1, \ldots, O_m)$ is defined as $\mathrm{FV}(O_1) \cup \ldots \cup \mathrm{FV}(O_m)$ when O_i is a formula, an expression, or a program.

Definition 4.1.12 *We define $Mod_1(P)$ for P in $\mathcal{L}^-$ and $Mod(P)$ for P in $\mathcal{L}$ as follows:*

$$
\begin{array}{lcl}
Mod_1(x := e) & = & \{x\}, \\
Mod_1(\mathit{if}\ (b)\ \mathit{then}\ (P_1)\ \mathit{else}\ \ (P_2)) & = & Mod_1(P_1) \cup Mod_1(P_2), \\
Mod_1(\mathit{while}\ (b)\ \mathit{do}\ (P)) & = & Mod_1(P), \\
Mod_1(P_1; P_2) & = & Mod_1(P_1) \cup Mod_1(P_2), \\
Mod_1(\mathit{skip}) & = & \emptyset, \\
Mod_1(x := \mathit{cons}(e_1, e_2)) & = & \{x\}, \\
Mod_1(x := [e]) & = & \{x\}, \\
Mod_1([e_1] := e_2) & = & \emptyset, \\
Mod_1(\mathit{dispose}(e)) & = & \emptyset, \\
Mod(P) & = & Mod_1(P^{(n_{proc})}).
\end{array}
$$

The expression $\mathrm{Mod}(P)$ is the set of variables that may be modified by P.

Lemma 4.1.13 *(1)* $FV(R_i^{(k+1)}) = \bigcup_{R_j \in PD(R_i,k)} FV(Q_j)$.

(2) $Mod_1(R_i^{(k+1)}) = \bigcup_{R_j \in PD(R_i,k)} Mod_1(Q_j)$.

Proof. (1) It is proved by induction on k.

Case 1. $k = 0$.

By definition, $\mathrm{FV}(R_i^{(1)}) = \mathrm{FV}(R_i[\overrightarrow{Q^{(0)}}]) = \mathrm{FV}(Q_i^{(0)}) = \mathrm{FV}(Q_i[\overrightarrow{\Omega}]) = \mathrm{FV}(Q_i)$. Since $\mathrm{PD}(R_i, 0) = \{R_i\}$, we have $\mathrm{FV}(Q_i) = \bigcup_{R_j \in \mathrm{PD}(R_i,0)} \mathrm{FV}(Q_j)$.

Case 2. k to be $k' + 1$.

By Proposition 4.1.10 (3), $\mathrm{FV}(R_i^{(k'+2)}) = \mathrm{FV}(Q_i[\overrightarrow{R^{(k'+1)}}])$. Then we have $\mathrm{FV}(R_i^{(k'+2)}) = \mathrm{FV}(Q_i) \cup \bigcup_{R_i \overset{1}{\leadsto} R_j} \mathrm{FV}(R_j^{(k'+1)})$. By induction hypothesis, we have $\mathrm{FV}(R_i^{(k'+2)}) = \mathrm{FV}(Q_i) \cup \bigcup_{R_i \overset{1}{\leadsto} R_j} \bigcup_{R_m \in \mathrm{PD}(R_j,k')} \mathrm{FV}(Q_m)$. Then we have $\mathrm{FV}(R_i^{(k'+2)}) = \mathrm{FV}(Q_i) \cup \bigcup_{R_m \in \mathrm{PD}(R_i,k'+1)} \mathrm{FV}(Q_m)$. Therefore, $\mathrm{FV}(R_i^{(k'+2)}) = \bigcup_{R_j \in \mathrm{PD}(R_i,k'+1)} (\mathrm{FV}(Q_j))$.

(2) Its proof is similar to (1). □

Proposition 4.1.14 *(1)* $FV(P^{(k)}) \subseteq EFV(P)$ *for all* k.

(2) $Mod_1(P^{(k)}) \subseteq Mod(P)$ *for all* k.

Proof. (1) Fix k. If $k = 0$, the claim trivially holds. Assume $k > 0$. By Lemma 4.1.6 (2), we have $\mathrm{PD}(R_i, k-1) \subseteq \mathrm{PD}(R_i, n_{proc} - 1)$. Then $\bigcup_{R_j \in \mathrm{PD}(R_i,k-1)} \mathrm{FV}(Q_j) \subseteq \bigcup_{R_j \in \mathrm{PD}(R_i,n_{proc}-1)} \mathrm{FV}(Q_j)$. Then by Proposition 4.1.13 (1), $\mathrm{FV}(R_i^{(k)}) \subseteq \mathrm{FV}(R_i^{(n_{proc})})$. Then we have $\mathrm{FV}(P) \cup \bigcup_{R_i \in \mathrm{PN}(P)} \mathrm{FV}(R_i^{(k)}) \subseteq \mathrm{FV}(P) \cup \bigcup_{R_i \in \mathrm{PN}(P)} \mathrm{FV}(R_i^{(n_{proc})})$. Then by Proposition 4.1.10 (1) we have $\mathrm{FV}(P^{(k)}) \subseteq \mathrm{FV}(P^{(n_{proc})})$. By definition, $\mathrm{FV}(P^{(k)}) \subseteq \mathrm{EFV}(P)$.

(2) Its proof is similar to (1). □

4.2 Semantics

The semantics of our programming language and our assertion language is defined in this section. Our semantics is based on the same structure as that in Reynolds' paper [18] except the following simplification: (1) values are natural numbers, (2) addresses are non-zero natural numbers, and (3) null is 0.

The set N is defined as the set of natural numbers. The set Vars is defined as the set of variables in the base language. The set Locs is defined as the set $\{n \in N | n > 0\}$.

For sets $S_1, S_2, f : S_1 \to S_2$ means that f is a function from S_1 to S_2. $f : S_1 \to_{fin} S_2$ means that f is a finite function from S_1 to S_2, that is, there is a finite subset S_1' of S_1 and $f : S_1' \to S_2$. $\mathrm{Dom}(f)$ denotes the domain of the function f. The expression $p(S)$ denotes the power set of the set S. For a function $f : A \to B$ and a subset $C \subseteq A$, the function $f|_C : C \to B$ is defined by $f|_C(x) = f(x)$ for $x \in C$.

A store is defined as a function from Vars $\to N$, and denoted by s. A heap is defined as a finite function from Locs $\to_{fin} N$, and denoted by h. A value is a natural number. An address is a positive natural number. The null pointer is 0. A store assigns a value to each variable. A heap assigns a value to an address in its finite domain.

The store $s[x_1 := n_1, \ldots, x_k := n_k]$ is defined by s' such that $s'(x_i) = n_i$ and $s'(y) = s(y)$ for $y \notin \{x_1, \ldots, x_k\}$. The heap $h[m_1 := n_1, \ldots, m_k := n_k]$ is defined by h' such that $h'(m_i) = n_i$ and $h'(y) = h(y)$ for $y \in \mathrm{Dom}(h) - \{m_1, \ldots, m_k\}$. The store $s[x_1 := n_1, \ldots, x_k := n_k]$ is the same as s except values for the variables $x_1, \ldots, x_k$. The heap $h[m_1 := n_1, \ldots, m_k := n_k]$ is the same as h except the contents of the memory cells at the addresses $m_1, \ldots, m_k$.

We will write $h = h_1 + h_2$ when $\mathrm{Dom}(h) = \mathrm{Dom}(h_1) \cup \mathrm{Dom}(h_2)$, $\mathrm{Dom}(h_1) \cap \mathrm{Dom}(h_2) = \emptyset$, $h(x) = h_1(x)$ for $x \in \mathrm{Dom}(h_1)$, and $h(x) = h_2(x)$ for $x \in \mathrm{Dom}(h_2)$. The heap h is divided into the two disjoint heaps h_1 and h_2 when $h = h_1 + h_2$.

A state is defined as (s, h). The set States is defined as the set of states. The state for a pointer program is specified by the store and the heap, since pointer programs manipulate memory heaps as well as variable assignments.

Definition 4.2.1 *We define the semantics of our base language by the standard model of natural numbers and* $[\![null]\!] = 0$. *That is, we suppose* $[\![0]\!] = 0$, $[\![1]\!] = 1$, $[\![+]\!] = +$, $[\![\times]\!] = \times$, $[\![=]\!] = (=)$, *and* $[\![<]\!] = (<)$. *For a store s, an expression e, and a pure formula A, according to the interpretation of a first-order language, the meaning* $[\![e]\!]_s$ *is defined as a natural number and the meaning* $[\![A]\!]_s$ *is defined as True or False.*

The expression $[\![e]\!]_s$ and $[\![A]\!]_s$ are the value of e under the store s, and the truth value of A under the store s, respectively.

4.2.1 Semantics of Programs

The relation $\subseteq$ over the functions of type States $\cup \{\text{abort}\} \to p(\text{States} \cup \{\text{abort}\})$ is necessary to define the semantics for while (b) do (P).

Definition 4.2.2 *We define* $\subseteq$ *for functions* F, G: *States* $\cup \{abort\} \to p(States \cup \{abort\})$. $F \subseteq G$ *is defined to hold if* $\forall r \in States\ (F(r) \subseteq G(r))$.

Definition 4.2.3 *We define the semantics of our programming language. For a program P, its meaning* $[\![P]\!]$ *is defined as a function from States* $\cup \{abort\}$ *to* $p(States \cup \{abort\})$. *We will define* $[\![P]\!](r_1)$ *as the set of all the possible resulting states after the execution of P terminates with the initial state* r_1. *In particular, if the execution of P with the initial state* r_1 *does not terminate, we will define* $[\![P]\!](r_1)$ *as the empty set* $\emptyset$. *The set* $[\![P]\!](\{r_1, \ldots, r_m\})$ *is defined as* $\bigcup_{i=1}^{m} [\![P]\!](r_i)$. *In order to define* $[\![P]\!]$ *we would like to define* $[\![P]\!]^-$ *for all P in the* *language $\mathcal{L}^-$. We define $[\![P]\!]^-$ by induction on P in $\mathcal{L}^-$ as follows:*

$$
\begin{aligned}
&[\![P]\!]^-(abort) = \{abort\},\\
&[\![x := e]\!]^-((s,h)) = \{(s[x := [\![e]\!]_s], h)\},\\
&[\![\mathit{if}\ (b)\ \mathit{then}\ (P_1)\ \mathit{else}\ (P_2)]\!]^-((s,h)) = \begin{cases} [\![P_1]\!]((s,h)) & \mathit{if}\ [\![b]\!]_s = \mathit{True},\\ [\![P_2]\!]((s,h)) & \mathit{otherwise},\end{cases}\\
&[\![\mathit{while}\ (b)\ \mathit{do}\ (P)]\!]^-\ \text{is the least function satisfying}\\
&\qquad [\![\mathit{while}\ (b)\ \mathit{do}\ (P)]\!]^-(abort) = \{abort\},\\
&\qquad [\![\mathit{while}\ (b)\ \mathit{do}\ (P)]\!]^-((s,h)) = \{(s,h)\} \qquad \mathit{if}\ [\![b]\!]_s = \mathit{False},\\
&\qquad [\![\mathit{while}\ (b)\ \mathit{do}\ (P)]\!]^-((s,h)) =\\
&\qquad\qquad \bigcup\{\ [\![\mathit{while}\ (b)\ \mathit{do}\ (P)]\!]^-(r) \mid r \in [\![P]\!]^-((s,h))\ \}\ \mathit{otherwise},\\
&[\![P_1; P_2]\!]^-((s,h)) = \bigcup\{\ [\![P_2]\!]^-(r) \mid r \in [\![P_1]\!]^-((s,h))\ \},\\
&[\![skip]\!]^-((s,h)) = \{(s,h)\},\\
&[\![x := cons(e_1, e_2)]\!]^-((s,h)) =\\
&\qquad \{(s[x := n], h[n := [\![e_1]\!]_s, n+1 := [\![e_2]\!]_s]) \mid n > 0, n, n+1 \notin Dom(h)\},\\
&[\![x := [e]]\!]^-((s,h)) \quad = \begin{cases} \{(s[x := h([\![e]\!]_s)], h)\} & \mathit{if}\ [\![e]\!]_s \in Dom(h),\\ \{abort\} & \mathit{otherwise},\end{cases}\\
&[\![[e_1] := e_2]\!]^-((s,h)) \quad = \begin{cases} \{(s, h[[\![e_1]\!]_s := [\![e_2]\!]_s])\} & \mathit{if}\ [\![e_1]\!]_s \in Dom(h),\\ \{abort\} & \mathit{otherwise},\end{cases}\\
&[\![dispose(e)]\!]^-((s,h)) \quad = \begin{cases} \{(s, h|_{Dom(h) - \{[\![e]\!]_s\}})\} & \mathit{if}\ [\![e]\!]_s \in Dom(h),\\ \{abort\} & \mathit{otherwise}.\end{cases}
\end{aligned}
$$

We present two propositions which together state that the semantics of a while (b) do (P) can be constructed by the semantics of P.

Proposition 4.2.4 *$r \in [\![\mathit{while}\ (b)\ \mathit{do}\ (P)]\!]^-((s,h))$ if and only if there exist $m \geq 0$, $r_0, \ldots, r_m$ such that $r_0 = (s,h)$, $r_m = r$, $[\![b]\!]_{r_i} = \mathit{True}$ and $r_{i+1} \in [\![P]\!]^-(r_i)$ for $0 \leq i < m$, and one of the following holds:*

(1) $r \neq abort$ and $[\![b]\!]_r = \mathit{False}$,

(2) $r = abort$,

where we write $[\![b]\!]_{(s',h')}$ *for* $[\![b]\!]_{s'}$.

Proof. First we will show the only-if-part. Let $F((s,h)) = \{ r \mid m \geq 0, r_0 = (s,h), r_m = r, r_i = (s_i,h_i), [\![b]\!]_{s_i} = \text{True}, r_{i+1} \in [\![P]\!]^-(r_i)\ (0 \leq \forall i < m), (r = (s_m,h_m) \wedge [\![b]\!]_{s_m} = \text{False}) \vee r = \text{abort} \}$ and $F(\text{abort}) = \{\text{abort}\}$. We will show that F satisfies the inequations obtained from the equations for $[\![\text{while } (b) \text{ do } (P)]\!]^-$ by replacing $=$ by $\supseteq$. That is, we will show

$$\begin{aligned}
&F(\text{abort}) \supseteq \{\text{abort}\},\\
&F((s,h)) \supseteq \{(s,h)\} \text{ if } [\![b]\!]_s = \text{False},\\
&F((s,h)) \supseteq \bigcup \{ F(r) \mid r \in [\![P]\!]^-((s,h)) \} \text{ if } [\![b]\!]_s = \text{True}.
\end{aligned}$$

By the well-known least fixed point theorem [22], these inequations imply our only-if-part.

The first inequation immediately holds by the definition of F. For the second inequation, since $[\![b]\!]_s = \text{False}$, by taking m to be 0, we have $(s,h) \in F((s,h))$. We will show the third inequation. Assume $[\![b]\!]_s = \text{True}$ and r is in the right-hand side. We will show $r \in F((s,h))$.

We have $q \in [\![P]\!]^-((s,h))$ and $r \in F(q)$.

Case 1. $q = (s'',h'')$.

By the definition of $r \in F(q)$, we have $m \geq 0, r_0 = (s'',h''), r_m = r, r_i = (s_i,h_i), [\![b]\!]_{s_i} = \text{True}, r_{i+1} \in [\![P]\!]^-(r_i)\ (0 \leq \forall i < m)$, and one of the following holds: either $r = (s_m,h_m)$ and $[\![b]\!]_{s_m} = \text{False}$, or $r = \text{abort}$.

Let $m' = m+1, r'_0 = (s,h)$, and $r'_i = r_{i-1}$ for $0 < i \leq m'$. By taking m and r_i to be m' and r'_i respectively in the definition of F, we have $r \in F((s,h))$.

Case 2. $q = \text{abort}$.

By the definition of F we have $r = \text{abort}$. By taking $m = 1$, we have $\text{abort} \in F((s,h))$.

Next we will show the if-part by induction on m. We assume the right-hand side.

We will show $r \in [\![\text{while } (b) \text{ do } (P)]\!]^-((s,h))$.

If $m = 0$ then we have $[\![b]\!]_s = \text{False}$ and $r = (s,h)$. Hence $r \in [\![\text{while } (b) \text{ do } (P)]\!]^-((s,h))$.

Suppose $m > 0$. We have m and $r_0, \ldots, r_m$ satisfying the conditions (1) or (2). If $r_1 = \text{abort}$, we have $m = 1$ and $\text{abort} \in [\![P]\!]^-((s,h))$. Hence $r \in [\![\text{while } (b) \text{ do } (P)]\!]^-((s,h))$ in this case. Suppose $r_1 \neq \text{abort}$. By induction hypothesis for $m-1$, we have $r \in [\![\text{while } (b) \text{ do } (P)]\!]^-(r_1)$. Since $[\![b]\!]_s = \text{True}$ and $r_1 \in [\![P]\!]^-((s,h))$, by the definition we have $r \in [\![\text{while } (b) \text{ do } (P)]\!]^-((s,h))$. $\square$

The following proposition characterizes the two properties of the semantics of Ω.

Proposition 4.2.5 *(1)* $[\![\Omega]\!]^-(\mathit{abort}) = \{\mathit{abort}\}$.

(2) $[\![\Omega]\!]^-((s,h)) = \emptyset$.

Proof. (1) By definition, $[\![\Omega]\!]^-(\text{abort}) = \{\text{abort}\}$.

(2) By definition, $[\![\Omega]\!]^-((s,h)) = [\![\text{while } (0=0) \text{ do } (\text{skip})]\!]^-((s,h))$. Since $[\![\text{skip}]\!]^-((s',h')) \not\ni \text{abort}$ for all s', h', by Proposition 4.2.4, $\text{abort} \notin [\![\text{while } (0=0) \text{ do } (\text{skip})]\!]^-((s,h))$. Since $[\![0=0]\!]_{s'} = \text{True}$ for all s', by Proposition 4.2.4 we have $(s',h') \notin [\![\text{while } (0=0) \text{ do } (\text{skip})]\!]^-((s,h))$ for all s', h'. Therefore, $[\![\text{while } (0=0) \text{ do } (\text{skip})]\!]^-((s,h)) = \emptyset$. $\square$

Lemma 4.2.6 *If $P'_i, P''_i \in \mathcal{L}^-$ $(1 \leq i \leq n_{proc})$ and $[\![P'_i]\!]^- \subseteq [\![P''_i]\!]^-$ for all i then $[\![P[\overrightarrow{P'}]]\!]^- \subseteq [\![P[\overrightarrow{P''}]]\!]^-$ where $P \in \mathcal{L}$ and $P[\overrightarrow{P'}] \in \mathcal{L}^-$.*

Proof. (1) By induction on P. Assume $[\![P'_i]\!]^- \subseteq [\![P''_i]\!]^-$ for all i.

Case 1. P is *atomic*.

Since $P[\overrightarrow{P'}]$=P=$P[\overrightarrow{P''}]$, the claim holds.

Case 2. P is if (b) then (P_1) else (P_2).

We will show that $[\![P[\overrightarrow{P'}]]\!]^-(r) \subseteq [\![P[\overrightarrow{P''}]]\!]^-(r)$.

Case 2.1. r is abort.

By definition, $[\![P[\overrightarrow{P'}]]\!]^-(\text{abort}) = \{\text{abort}\} = [\![P[\overrightarrow{P''}]]\!]^-(\text{abort})$. Hence $[\![P[\overrightarrow{P'}]]\!]^- \subseteq [\![P[\overrightarrow{P''}]]\!]^-$.

Case 2.2. r is (s,h).

Let r to be (s,h). Suppose $[\![b]\!]_s = \text{True}$. Then by definition, $[\![P[\overrightarrow{P'}]]\!]^-(r) = [\![P_1[\overrightarrow{P'}]]\!]^-(r)$ and $[\![P[\overrightarrow{P''}]]\!]^-(r) = [\![P_1[\overrightarrow{P''}]]\!]^-(r)$. By induction hypothesis, $[\![P_1[\overrightarrow{P'}]]\!]^- \subseteq [\![P_1[\overrightarrow{P''}]]\!]^-$. Therefore, $[\![P[\overrightarrow{P'}]]\!]^- \subseteq [\![P[\overrightarrow{P''}]]\!]^-$. In the case $[\![b]\!]_s = \text{False}$, it can be proved similarly.

Case 3. P is while (b) do (P_1).

We will show that $[\![P[\overrightarrow{P'}]]\!]^-(r) \subseteq [\![P[\overrightarrow{P''}]]\!]^-(r)$.

Case 3.1. r is abort.

The case is similar to 2.1.

Case 3.2. r to be (s,h).

Case $[\![b]\!]_s = \text{False}$. By definition, $[\![\text{while } (b) \text{ do } (P_1[\overrightarrow{P'}])]\!]^-((s,h)) = \{(s,h)\} = [\![\text{while } (b) \text{ do } (P_1[\overrightarrow{P''}])]\!]^-((s,h))$. Hence $[\![P[\overrightarrow{P'}]]\!]^- \subseteq [\![P[\overrightarrow{P''}]]\!]^-$.

Case $[\![b]\!]_s = \text{True}$. Assume $[\![\text{while } (b) \text{ do } (P_1[\overrightarrow{P'}])]\!]^-((s,h)) \ni r'$. By Proposition 4.2.4, we have $m \geq 0, r_0, \ldots, r_m$ such that $r_0 = (s,h), r_{i+1} \in [\![P_1[\overrightarrow{P'}]]\!]^-(r_i)$ and $[\![b]\!]_{r_i} = \text{True}$ for $0 \leq i < m$ such that either $r' \neq \text{abort}$ and $[\![b]\!]_{r'} = \text{False}$, or $r' = \text{abort}$. By induction hypothesis, $r_{i+1} \in [\![P_1[\overrightarrow{P''}]]\!]^-(r_i)$. Then $[\![\text{while } (b) \text{ do } (P_1[\overrightarrow{P''}])]\!]^-(r) \ni r'$.

Then $[\![\text{while } (b) \text{ do } (P_1[\overrightarrow{P'}])]\!]^-(r) \subseteq [\![\text{while } (b) \text{ do } (P_1[\overrightarrow{P''}])]\!]^-(r)$. Therefore, $[\![P[\overrightarrow{P'}]]\!]^- \subseteq [\![P[\overrightarrow{P''}]]\!]^-$.

Case 4. P is $P_1; P_2$.

By induction hypothesis, $[\![P_1[\overrightarrow{P'}]]\!]^- \subseteq [\![P_1[\overrightarrow{P''}]]\!]^-$ and $[\![P_2[\overrightarrow{P'}]]\!]^- \subseteq [\![P_2[\overrightarrow{P''}]]\!]^-$. Therefore, $[\![P[\overrightarrow{P'}]]\!]^- \subseteq [\![P[\overrightarrow{P''}]]\!]^-$.

Case 5. R_i.

We have $R_i[\overrightarrow{P'}] = P'_i$ and $R_i[\overrightarrow{P''}] = P''_i$. Hence $[\![R_i[\overrightarrow{P'}]]\!]^- \subseteq [\![R_i[\overrightarrow{P''}]]\!]^-$. $\square$

We have already defined the semantics of P in the language $\mathcal{L}^-$. We define the semantics of P in $\mathcal{L}$. The semantics we will define is usually called an approximating semantics.

Definition 4.2.7 *The semantics of P in $\mathcal{L}$ is defined by* $[\![P]\!](r) = \bigcup_{i=0}^{\infty}([\![P^{(i)}]\!]^-(r))$.

Note that $Q_i^{(k)}$ is a program of the language $\mathcal{L}^-$ and doesn't contain R_i.

Remark that for P in $\mathcal{L}^-$, we have $[\![P]\!] = [\![P]\!]^-$ since $P^{(k)} = P$.

Lemma 4.2.8 *For all k,* $[\![P^{(k)}]\!]^- \subseteq [\![P^{(k+1)}]\!]^-$.

Proof. By induction on k.

Case 1. $k = 0$.

We have $[\![\Omega]\!]^-(r) = \emptyset$ for all r by Proposition 4.2.5 (2). Then for all i, we have $[\![\Omega]\!]^- \subseteq [\![Q_i^{(0)}]\!]^-$. Then by Lemma 4.2.6, $[\![P[\overrightarrow{\Omega}]]\!] \subseteq [\![P[\overrightarrow{Q^{(0)}}]]\!]$. Then $[\![P^{(0)}]\!]^- \subseteq [\![P^{(1)}]\!]^-$.

Case 2. $k > 0$.

Let k be $k' + 1$. We have $Q^{(k)} = Q^{(k'+1)} = Q[\overrightarrow{Q^{(k')}}]$ and $Q^{(k+1)} = Q^{(k'+2)} = Q[\overrightarrow{Q^{(k'+1)}}]$. By induction hypothesis, $[\![Q^{(k')}]\!]^- \subseteq [\![Q^{(k'+1)}]\!]^-$. By Lemma 4.2.6, we now have $[\![P[\overrightarrow{Q^{(k')}}]]\!]^- \subseteq [\![P[\overrightarrow{Q^{(k'+1)}}]]\!]^-$. Then $[\![P^{(k)}]\!]^- \subseteq [\![P^{(k+1)}]\!]^-$. □

4.2.2 Semantics of Assertions

Definition 4.2.9 *We define the semantics of the assertion language. For an assertion A and a state (s, h), the meaning $[\![A]\!]_{(s,h)}$ is defined as True or False. $[\![A]\!]_{(s,h)}$ is the truth value* *of A at the state (s,h). $[\![A]\!]_{(s,h)}$ is defined by induction on A as follows:*

$$
\begin{array}{lcll}
[\![emp]\!]_{(s,h)} & = & \mathit{True\ if\ Dom}(h) = \emptyset, & \\
[\![e_1 = e_2]\!]_{(s,h)} & = & ([\![e_1]\!]_s = [\![e_2]\!]_s), & \\
[\![e_1 < e_2]\!]_{(s,h)} & = & ([\![e_1]\!]_s < [\![e_2]\!]_s), & \\
[\![e_1 \mapsto e_2]\!]_{(s,h)} & = & \mathit{True} & \mathit{if\ Dom}(h) = \{[\![e_1]\!]_s\} \mathit{\ and\ } h([\![e_1]\!]_s) = [\![e_2]\!]_s, \\
[\![\neg A]\!]_{(s,h)} & = & (\mathit{not}\ [\![A]\!]_{(s,h)}), & \\
[\![A \wedge B]\!]_{(s,h)} & = & ([\![A]\!]_{(s,h)}\ \mathit{and}\ [\![B]\!]_{(s,h)}), & \\
[\![A \vee B]\!]_{(s,h)} & = & ([\![A]\!]_{(s,h)}\ \mathit{or}\ [\![B]\!]_{(s,h)}), & \\
[\![A \to B]\!]_{(s,h)} & = & ([\![A]\!]_{(s,h)}\ \mathit{implies}\ [\![B]\!]_{(s,h)}), & \\
[\![\forall x A]\!]_{(s,h)} & = & \mathit{True} & \mathit{if}\ [\![A]\!]_{(s[x:=m],h)} = \mathit{True\ for\ all}\ m \in N, \\
[\![\exists x A]\!]_{(s,h)} & = & \mathit{True} & \mathit{if}\ [\![A]\!]_{(s[x:=m],h)} = \mathit{True\ for\ some}\ m \in N, \\
[\![A * B]\!]_{(s,h)} & = & \mathit{True} & \mathit{if}\ h = h_1 + h_2\ \mathit{and} \\
 & & & [\![A]\!]_{(s,h_1)} = [\![B]\!]_{(s,h_2)} = \mathit{True\ for\ some}\ h_1, h_2, \\
[\![A -\!\!* B]\!]_{(s,h)} & = & \mathit{True} & \mathit{if}\ h_2 = h_1 + h\ \mathit{and} \\
 & & & [\![A]\!]_{(s,h_1)} = \mathit{True\ implies}\ [\![B]\!]_{(s,h_2)} = \mathit{True\ for\ all}\ h_1, h_2.
\end{array}
$$

We say A is true when $[\![A]\!]_{(s,h)} = True$ for all (s,h).

4.2.3 Semantics of Asserted Program

Finally we need to define the semantics of the asserted programs. It is basically the same as in [18].

Definition 4.2.10 *For an asserted program $\{A\}P\{B\}$, the meaning of $\{A\}P\{B\}$ is defined as True or False. $\{A\}P\{B\}$ is defined to be True if both of the following hold.*

(1) for all (s,h), if $[\![A]\!]_{(s,h)} = True$, then $[\![P]\!]((s,h)) \not\ni abort$.

(2) for all (s,h) and (s',h'), if $[\![A]\!]_{(s,h)} = True$ and $[\![P]\!]((s,h)) \ni (s',h')$, then $[\![B]\!]_{(s',h')} = True$.

Remark that the semantics of an asserted program with a non-terminating program is always True. Because, according to the definition of semantics, a resulting abort state of a program implies termination of its execution.

In our system, judgments are in the form $\Gamma \vdash \{A\}P\{B\}$ where Γ is a set of asserted programs. Here we will define semantics of a judgment.

We say Γ is true if all $\{A\}P\{B\}$ in Γ are true.

Definition 4.2.11 *We say $\Gamma \vdash \{A\}P\{B\}$ is true when the following holds: $\{A\}P\{B\}$ is true if $\{A_i\}P_i\{B_i\}$ is true for all $\{A_i\}P_i\{B_i\} \in \Gamma$.*

We say Γ is true if all $\{A\}P\{B\}$ in Γ are true.

The asserted program $\{A\}P\{B\}$ means abort-free partial correctness and also implies partial correctness in the standard sense. Namely, it means that the execution of the program P with all the initial states which satisfy A never aborts, that is, P does not access any of the unallocated addresses during the execution. It is one of the strongest feature of the system.

The next lemma shows that the semantics of a procedure call (or procedure name) is the same as that of its body.

Lemma 4.2.12 $[\![R_i]\!] = [\![Q_i]\!]$.

Proof. By definition we have $[\![Q_i]\!](r) = \bigcup_{k=0}^{\infty}([\![Q_i^{(k)}]\!]^{-}(r))$. Since $Q_i^{(k)} = R_i[\overrightarrow{Q^{(k)}}] = R_i^{(k+1)}$, we have $\bigcup_{k=0}^{\infty}([\![Q_i^{(k)}]\!]^{-}(r)) = \bigcup_{k=0}^{\infty}([\![R_i^{(k+1)}]\!]^{-}(r))$. Then by Proposition 4.2.5 (2), $[\![Q_i]\!](r) = \bigcup_{k=0}^{\infty}([\![R_i^{(k+1)}]\!]^{-}(r)) \cup [\![\Omega]\!]^{-}(r)$. Then $[\![Q_i]\!](r) = \bigcup_{k=0}^{\infty}([\![R_i^{(k+1)}]\!]^{-}(r)) \cup [\![R_i^{(0)}]\!]^{-}(r)$ by Proposition 4.1.10(2). Then $[\![Q_i]\!](r) = \bigcup_{k=0}^{\infty}([\![R_i^{(k)}]\!]^{-}(r))$. Therefore, $[\![Q_i]\!] = [\![R_i]\!]$. □

We define the equality of two stores over a set of variables. Suppose some stores are equal over the set of free variables of a program. If we execute the program with those stores in the states, then the stores in the resulting states are still equal over the same set of free variables.

Definition 4.2.13 *For a set V of variables, we define $=_V$ as follows: $s =_V s'$ if and only if $s(x) = s'(x)$ for all $x \in V$.*

Definition 4.2.14 *We first define $[s_1, s_2, V]$ to denote the store s such that $s =_V s_1$ and $s =_{V^c} s_2$ for stores s_1, s_2 and a set of variables V.*

Lemma 4.2.15 *Suppose* $P \in \mathcal{L}^-$.

(1) If $[\![P]\!]^-((s,h)) \ni (s_1,h_1)$ *then* $s_1 =_{Mod_1(P)^c} s$.

(2) If $s =_{FV(P)} s'$, $[\![P]\!]^-((s,h)) \ni (s_1,h_1)$ *and* $s'_1 = [s_1, s', FV(P)]$ *then* $[\![P]\!]^-((s',h)) \ni (s'_1,h_1)$.

(3) If $s =_{FV(P)} s'$ *and* $[\![P]\!]^-((s,h)) \ni abort$*, then* $[\![P]\!]^-((s',h)) \ni abort$*.*

Proof. (1) We will show the claim by induction on P. We consider cases according to P.

Case 1. $x := e$.

Assume $[\![x := e]\!]^-((s,h)) \ni (s_1,h_1)$. By definition, $s_1 = s[x := [\![e]\!]_s]$ and $h = h_1$. Then for all $y \neq x$, $s(y) = s_1(y)$. By definition, $\mathrm{Mod}_1(x := e) = \{x\}$. Therefore, $s_1 =_{\mathrm{Mod}_1(P)^c} s$.

Case 2. P is if (b) then (P_1) else (P_2).

Assume $[\![P]\!]^-((s,h)) \ni (s_1,h_1)$.

Case $[\![b]\!]_s =$ True. By definition, we have $[\![P_1]\!]^-((s,h)) \ni (s_1,h_1)$. By induction hypothesis, $s =_{\mathrm{Mod}_1(P_1)^c} s_1$. We have $\mathrm{Mod}_1(P)^c \subseteq \mathrm{Mod}_1(P_1)^c$. Therefore, $s =_{\mathrm{Mod}_1(P)^c} s_1$.

Case $[\![b]\!]_s =$ False can be shown as above.

Case 3. P is while (b) do (P_1).

Assume $[\![P]\!]^-((s,h)) \ni (s_1,h_1)$.

Case $[\![b]\!]_s =$ True. By Proposition 4.2.4, we have $s_2,\ldots,s_m,h_2,\ldots,h_m$ such that $(s,h) = (s_2,h_2)$, $(s_1,h_1) = (s_m,h_m)$, for all $i = 2,\ldots,m-1$, $[\![P_1]\!]^-((s_i,h_i)) \ni (s_{i+1},h_{i+1})$, $[\![b]\!]_{s_i} =$ True and $[\![b]\!]_{s_m} =$ False. By induction hypothesis, for all $i = 2,\ldots,m-1$, $s_i =_{\mathrm{Mod}_1(P_1)^c} s_{i+1}$. Since $\mathrm{Mod}_1(P)^c \subseteq \mathrm{Mod}_1(P_1)^c$, for all $i = 2,\ldots,m-1$, $s_i =_{\mathrm{Mod}_1(P)^c} s_{i+1}$. Therefore, $s =_{\mathrm{Mod}_1(P)^c} s_1$.

Case $[\![b]\!]_s =$ False. Then by definition, $s = s_1$. Then $s =_{\mathrm{Mod}_1(P)^c} s_1$.

Case 4. $P_1; P_2$.

Assume $[\![P_1; P_2]\!]^-((s,h)) \ni (s_1,h_1)$. By definition, we have s_2, h_2 such that $[\![P_1]\!]^-((s,h)) \ni (s_2,h_2)$ and $[\![P_2]\!]^-((s_2,h_2)) \ni (s_1,h_1)$.

By induction hypothesis, $s_2 =_{\mathrm{Mod}_1(P_1)^c} s$ and $s_1 =_{\mathrm{Mod}_1(P_2)^c} s_2$. Since $\mathrm{Mod}_1(P)^c \subseteq \mathrm{Mod}_1(P_1)^c$ and $\mathrm{Mod}_1(P)^c \subseteq \mathrm{Mod}_1(P_2)^c$, we have $s_2 =_{\mathrm{Mod}_1(P)^c} s$ and $s_1 =_{\mathrm{Mod}_1(P)^c} s_2$. Therefore, $s_1 =_{\mathrm{Mod}_1(P)^c} s$.

Case 5. skip.

Its proof is immediate.

Case 6. $x := \mathrm{cons}(e_1, e_2)$.

Assume $[\![x := \mathrm{cons}(e_1,e_2)]\!]^-((s,h)) \ni (s_1,h_1)$. By definition, (s_1,h_1) is in $\{\, (s[x := m],\, h[m := [\![e_1]\!]_s, m+1 := [\![e_2]\!]_s]) \mid m > 0,\ m,\ m+1 \notin \mathrm{Dom}(h) \,\}$. So, for all $y \neq x$, $s(y) = s_1(y)$. By definition, $\mathrm{Mod}_1(x := \mathrm{cons}(e_1,e_2)) = \{x\}$. Therefore, $s =_{\mathrm{Mod}_1(x:=\mathrm{cons}(e_1,e_2))^c} s_1$.

Case 7. $x := [e]$.

Assume $[\![x := [e]]\!]^-((s,h)) \ni (s_1,h_1)$. By definition, $\{(s[x := h([\![e]\!]_s)], h)\}$. So, for all $y \neq x$, $s(y) = s_1(y)$. By definition, $\mathrm{Mod}_1(x := [e]) = \{x\}$. Therefore, $s =_{\mathrm{Mod}_1(x:=[e])^c} s_1$.

Case 8. $[e_1] := e_2$.

Assume $[\![[e_1] := e_2]\!]^-((s,h)) \ni (s_1,h_1)$. By definition, $s = s_1$. Therefore, $s =_{\mathrm{Mod}_1([e_1]:=e_2)^c} s_1$.

Case 9. $\mathrm{dispose}(e)$.

Assume $[\![\mathrm{dispose}(e)]\!]^-((s,h)) \ni (s_1,h_1)$. By definition, $s = s_1$. Therefore, $s =_{\mathrm{Mod}_1(\mathrm{dispose}(e))^c} s_1$.

(2) We will show the claim by induction on P. We consider cases according to P.

Case 1. P is $x := e$.

Assume $s =_{\mathrm{FV}(x:=e)} s'$, $[\![x := e]\!]^-((s,h)) \ni (s_1,h_1)$ and $s_1' = [s_1, s', \mathrm{FV}(x := e)]$. Then by definition, $s_1 = s[x := [\![e]\!]_s]$ and $s_1' =_{\mathrm{FV}(x:=e)} s_1$. Then $s_1'(x) = s_1(x) = [\![e]\!]_s =$

$[\![e]\!]_{s'}$. We have $s'_1 =_{\mathrm{FV}(x:=e)^c} s'$ and for all $y \in \mathrm{FV}(e)$ and $y \neq x$, $s'_1(y) = s_1(y) = s(y) = s'(y)$. Then we have $s'_1 = s'[x := [\![e]\!]_{s'}]$. Therefore, $[\![x := e]\!]^-((s', h)) \ni (s'_1, h_1)$.

Case 2. P is if (b) then (P_1) else (P_2).

Assume $s =_{\mathrm{FV}(\text{if }(b)\text{ then }(P_1)\text{ else }(P_2))} s'$, $[\![\text{if }(b)\text{ then }(P_1)\text{ else }(P_2)]\!]^-((s, h)) \ni (s_1, h_1)$ and $s'_1 = [s_1, s', \mathrm{FV}(\text{if }(b)\text{ then }(P_1)\text{ else }(P_2))]$.

Let $[\![b]\!]_s = \mathrm{True}$. Then by definition $[\![P_1]\!]^-((s, h)) \ni (s_1, h_1)$. By (1), $s =_{\mathrm{Mod}_1(P_1)^c} s_1$ and since $\mathrm{FV}(P_1)^c \subseteq \mathrm{Mod}_1(P_1)^c$, we have $s =_{\mathrm{FV}(P_1)^c} s_1$. We also have $s =_{\mathrm{FV}(P)} s'$ by assumption and $s'_1 =_{\mathrm{FV}(P)} s_1$ by definition and hence $s' =_{\mathrm{FV}(P_1)} s$ and $s'_1 =_{\mathrm{FV}(P_1)} s_1$ because $\mathrm{FV}(P_1) \subseteq \mathrm{FV}(P)$. Then for all $y \in \mathrm{FV}(P) - \mathrm{FV}(P_1)$, $s'(y) = s(y) = s_1(y) = s'_1(y)$. Since $s'_1 =_{\mathrm{FV}(P)-\mathrm{FV}(P_1)} s'$ and $s'_1 =_{\mathrm{FV}(P)^c} s'$ are true, we have that $s' =_{\mathrm{FV}(P_1)^c} s'_1$ holds. Then $s'_1 = [s_1, s', \mathrm{FV}(P_1)]$. By induction hypothesis, $[\![P_1]\!]^-((s', h)) \ni (s'_1, h_1)$. Then by definition $[\![P]\!]^-((s', h)) \ni (s'_1, h_1)$.

Again let $[\![b]\!]_s = \mathrm{False}$. In the same way as above we can prove that $[\![P]\!]^-((s', h)) \ni (s'_1, h_1)$.

Case 3. P is while (b) do (P_1).

Assume $s =_{\mathrm{FV}(\text{while }(b)\text{ do }(P_1))} s'$, $[\![\text{while }(b)\text{ do }(P_1)]\!]^-((s, h)) \ni (s_1, h_1)$ and $s'_1 = [s_1, s', \mathrm{FV}(\text{while }(b)\text{ do }(P_1))]$. We have $(s, h) = (q_0, h'_0), \ldots, (q_m, h'_m) = (s_1, h_1)$ such that $[\![P_1]\!]^-((q_i, h'_i)) \ni (q_{i+1}, h'_{i+1})$ and $[\![b]\!]_{q_i} = \mathrm{True}$ for all $0 \leq i < m$ and $[\![b]\!]_{q_m} = \mathrm{False}$ by Proposition 4.2.4. Let $q'_i = [q_i, s', \mathrm{FV}(P_1)]$ for all $0 \leq i \leq m$.

We will show that $s' = q'_0$ and $s'_1 = q'_m$. We have $q'_0 = [s, s', \mathrm{FV}(P_1)]$. Then $q'_0 =_{\mathrm{FV}(P_1)^c} s'$. Since $s =_{\mathrm{FV}(P_1)} s'$ and $q'_0 =_{\mathrm{FV}(P_1)} s$, we have $q'_0 =_{\mathrm{FV}(P_1)} s'$. Therefore, $s' = q'_0$. We have $q'_m = [s_1, s', \mathrm{FV}(P_1)]$. We also have $s'_1 = [s_1, s', \mathrm{FV}(P)]$. Then $q'_m =_{\mathrm{FV}(P_1)} s'_1$. By (1), $s =_{\mathrm{Mod}_1(P)^c} s_1$. Then $s =_{\mathrm{FV}(P_1)^c} s_1$ since $\mathrm{Mod}_1(P)^c = \mathrm{Mod}_1(P_1)^c$ and $\mathrm{Mod}_1(P_1)^c \supseteq \mathrm{FV}(P_1)^c$. We have $q'_m =_{\mathrm{FV}(P)^c} s'$ since $\mathrm{FV}(P)^c \subseteq \mathrm{FV}(P_1)^c$ and $q'_m =_{\mathrm{FV}(P_1)^c} s'$. Then $q'_m =_{\mathrm{FV}(P)^c} s'_1$. Since $q'_m =_{\mathrm{FV}(P_1)^c} s'$, $s' =_{\mathrm{FV}(P)} s$ and $s_1 =_{\mathrm{FV}(P)} s'_1$, for all $y \in \mathrm{FV}(P) - \mathrm{FV}(P_1)$, $q'_m(y) = s'(y) = s(y) = s_1(y) = s'_1(y)$. Therefore, $s'_1 = q'_m$. Hence $s' = q'_0, s'_1 = q'_m$.

Since $q'_i =_{\mathrm{FV}(P_1)^c} s'$ by definition, $q'_{i+1} = [q_{i+1}, q'_i, \mathrm{FV}(P_1)]$ for $0 \leq i < m$. By induction hypothesis, $[\![P_1]\!]^-((q'_i, h_i)) \ni (q'_{i+1}, h_{i+1})$. We also have $[\![b]\!]_{q'_i} =$

True for $0 \leq i < m$ and $[\![b]\!]_{q'_m} = \text{False}$. Hence by Proposition 4.2.4, $[\![\text{while } (b) \text{ do } (P_1)]\!]^-((s', h)) \ni (s'_1, h_1)$.

Case 4. P is $P_1; P_2$.

Assume $s =_{\mathrm{FV}(P_1;P_2)} s'$, $[\![P_1; P_2]\!]^-((s, h)) \ni (s_1, h_1)$ and $s'_1 = [s_1, s', \mathrm{FV}(P_1; P_2)]$. By definition, we know that $[\![P_1]\!]^-((s, h)) \ni (s_2, h_2)$ and $[\![P_2]\!]^-((s_2, h_2)) \ni (s_1, h_1)$ for some s_2, h_2. By (1), $s =_{\mathrm{Mod}_1(P_1)^c} s_2$ and $s_1 =_{\mathrm{Mod}_1(P_2)^c} s_2$ and then $s =_{\mathrm{FV}(P_1)^c} s_2$ and $s_1 =_{\mathrm{FV}(P_2)^c} s_2$. Let take s'_2 such that $s'_2 = [s_2, s', \mathrm{FV}(P_1)]$. Then $s'_2 =_{\mathrm{FV}(P_1)} s_2$ and $s'_2 =_{\mathrm{FV}(P_1)^c} s'$. Then for all $y \in \mathrm{FV}(P_1) \cap \mathrm{FV}(P_2)$, $s'_2(y) = s_2(y)$. For all $y \in \mathrm{FV}(P_2) - \mathrm{FV}(P_1)$, $s'_2(y) = s'(y) = s(y) = s_2(y)$. So, $s'_2 =_{\mathrm{FV}(P_2)} s_2$. Since $s'_1 =_{\mathrm{FV}(P_1;P_2)} s_1$ we have $s'_1 =_{\mathrm{FV}(P_2)} s_1$. For all $y \in \mathrm{FV}(P_1) - \mathrm{FV}(P_2)$, $s'_1(y) = s_1(y)$ by $s'_1 = [s_1, s', \mathrm{FV}(P_1; P_2)]$, $s_1(y) = s_2(y)$ by $s_1 =_{\mathrm{FV}(P_2)^c} s_2$, $s_2(y) = s'_2(y)$ by $s'_2 =_{\mathrm{FV}(P_1)} s_2$ and hence $s'_1(y) = s'_2(y)$. For all $y \notin \mathrm{FV}(P_1; P_2)$, $s'_1(y) = s'(y)$ by $s'_1 = [s_1, s', \mathrm{FV}(P_1; P_2)]$ and $s'(y) = s'_2(y)$ by $s'_2 = [s_2, s', \mathrm{FV}(P_1)]$ and hence $s'_2(y) = s'_1(y)$. Then we have $s'_2 =_{\mathrm{FV}(P_2)^c} s'_1$. Then $s'_1 = [s_1, s'_2, \mathrm{FV}(P_2)]$.

Hence, by induction hypothesis $[\![P_1]\!]^-((s', h)) \ni (s'_2, h_2)$ and $[\![P_2]\!]^-((s'_2, h_2)) \ni (s'_1, h_1)$. Therefore, by definition $[\![P_1; P_2]\!]^-((s', h)) \ni (s'_1, h_1)$.

Case 5. P is skip.

Its proof is immediate.

Case 6. P is $x := \mathrm{cons}(e_1, e_2)$.

Assume $s =_{\mathrm{FV}(x:=\mathrm{cons}(e_1,e_2))} s'$, $[\![x := \mathrm{cons}(e_1, e_2)]\!]^-((s, h)) \ni (s_1, h_1)$ and $s'_1 = [s_1, s', \mathrm{FV}(x := \mathrm{cons}(e_1, e_2))]$. Then by definition, $s_1 = s[x := m]$ for some $m > 0$ where $m, m + 1 \notin \mathrm{Dom}(h)$ and $s'_1 =_{\mathrm{FV}(x:=\mathrm{cons}(e_1,e_2))} s_1$. Then $s'_1(x) = s_1(x) = m$. We have $s'_1 =_{\mathrm{FV}(x:=\mathrm{cons}(e_1,e_2))^c} s'$. For all $y \in \mathrm{FV}(x := \mathrm{cons}(e_1, e_2)) - \{x\}$, $s'_1(y) = s_1(y) = s(y) = s'(y)$. Then we have $s'_1 = s'[x := m]$. Therefore, by definition $[\![x := \mathrm{cons}(e_1, e_2)]\!]^-((s', h)) \ni (s'_1, h_1)$.

Case 7. P is $x := [e]$.

Assume $s =_{\mathrm{FV}(x:=[e])} s'$, $[\![x := [e]]\!]^-((s, h)) \ni (s_1, h_1)$ and $s'_1 = [s_1, s', \mathrm{FV}(x := [e])]$. Then by definition, $s_1 = s[x := h([\![e]\!]_s)]$ and $s'_1 =_{\mathrm{FV}(x:=[e])} s_1$. Then $s'_1(x) =$

$s_1(x) = h(\llbracket e \rrbracket_s) = h(\llbracket e \rrbracket_{s'})$. We have $s_1' =_{\mathrm{FV}(x:=[e])^c} s'$ and hence for all $y \neq x$ and $y \in \mathrm{FV}(x := [e])$, $s_1'(y) = s_1(y) = s(y) = s'(y)$. Then we have $s_1' = s'[x := h(\llbracket e \rrbracket_{s'})]$. Therefore, $\llbracket x := [e] \rrbracket^-((s', h)) \ni (s_1', h_1)$.

Case 8. P is $[e_1] := e_2$.

Assume $s =_{\mathrm{FV}([e_1]:=e_2)} s'$, $\llbracket [e_1] := e_2 \rrbracket^-((s, h)) \ni (s_1, h_1)$ and $s_1' = [s_1, s', \mathrm{FV}([e_1] := e_2)]$. Then by definition, $s_1 = s$ and $s_1' =_{\mathrm{FV}([e_1]:=e_2)} s_1$. For all $y \in \mathrm{FV}([e_1] := e_2)$, we have $s_1'(y) = s_1(y) = s(y) = s'(y)$. Then we have $s_1' = s'$. Since $h_1 = h[\llbracket e_1 \rrbracket_s := \llbracket e_2 \rrbracket_s]$, we have $h_1 = h[\llbracket e_1 \rrbracket_{s'} := \llbracket e_2 \rrbracket_{s'}]$. Therefore, $\llbracket [e_1] := e_2 \rrbracket^-((s', h)) \ni (s_1', h_1)$.

Case 9. P is $\mathrm{dispose}(e)$.

Assume $s =_{\mathrm{FV}(\mathrm{dispose}(e))} s'$, $\llbracket \mathrm{dispose}(e) \rrbracket^-((s, h)) \ni (s_1, h_1)$ and $s_1' = [s_1, s', \mathrm{FV}(\mathrm{dispose}(e))]$. Then by definition, $s_1 = s$ and $s_1' =_{\mathrm{FV}(\mathrm{dispose}(e))} s_1$. For all $y \in \mathrm{FV}(\mathrm{dispose}(e))$, we have $s_1'(y) = s_1(y) = s(y) = s'(y)$. Then we have $s_1' = s'$. Since $h_1 = h|_{\mathrm{Dom}(h) - \llbracket e \rrbracket_s}$, we have $h_1 = h|_{\mathrm{Dom}(h) - \llbracket e \rrbracket_{s'}}$. Therefore, $\llbracket \mathrm{dispose}(e) \rrbracket^-((s', h)) \ni (s_1', h_1)$.

(3) We will show the claim by induction on P. We consider cases according to P.

Case 1. $x := e$.

Assume $s =_{\mathrm{FV}(x:=e)} s'$ and $\llbracket x := e \rrbracket^-((s, h)) \ni \mathrm{abort}$. By definition, $\llbracket x := e \rrbracket^-((s, h)) = \{(s[x := \llbracket e \rrbracket_s], h)\}$. Therefore, $\llbracket x := e \rrbracket^-((s, h)) \ni \mathrm{abort}$ is False. Then $\llbracket x := e \rrbracket^-((s, h)) \ni \mathrm{abort}$ implies $\llbracket x := e \rrbracket^-((s', h)) \ni \mathrm{abort}$.

Case 2. P is if (b) then (P_1) else (P_2).

Assume $s =_{\mathrm{FV}(P)} s'$.

Case 2.1. $\llbracket b \rrbracket_s = \mathrm{True}$. Then $\llbracket b \rrbracket_{s'} = \mathrm{True}$. Assume $\llbracket P \rrbracket^-((s, h)) \ni \mathrm{abort}$. By definition, $\llbracket P_1 \rrbracket((s, h)) \ni \mathrm{abort}$. By definition $\mathrm{FV}(P_1) \subseteq \mathrm{FV}(P)$ and then $s =_{\mathrm{FV}(P_1)} s'$. By induction hypothesis, $\llbracket P_1 \rrbracket^-((s', h)) \ni \mathrm{abort}$. Therefore, by definition $\llbracket P \rrbracket^-((s', h)) \ni \mathrm{abort}$.

Case 2.2. $\llbracket b \rrbracket_s = \mathrm{False}$ can be proved similarly.

Case 3. P is while (b) do (P_1).

Assume $s =_{\mathrm{FV}(\text{while }(b)\text{ do }(P_1))} s'$.

Case 3.1. $[\![b]\!]_s = \text{True}$. Then $[\![b]\!]_{s'} = \text{True}$. Assume $[\![\text{while }(b)\text{ do }(P_1)]\!]((s,h)) \ni \text{abort}$. By Proposition 4.2.4, we have $s_1, \ldots, s_m, h_1, \ldots, h_m$ such that $(s,h) = (s_1,h_1)$, $[\![P_1]\!]^-((s_i,h_i)) \ni (s_{i+1},h_{i+1})$ and $[\![b]\!]_{s_i} = \text{True}$ for all $i = 1,\ldots,m-1$, $[\![P_1]\!]^-((s_m,h_m)) \ni \text{abort}$ and $[\![b]\!]_{s_m} = \text{True}$. By definition $\mathrm{FV}(P_1) \subseteq \mathrm{FV}(\text{while }(b)\text{ do }(P_1))$ and then $s =_{\mathrm{FV}(P_1)} s'$. Let $s'_i = [s_i, s', \mathrm{FV}(P)]$ for all $i = 1,\ldots,m$. Then $s'_1 =_{\mathrm{FV}(P)^c} s'$ and $s'_1 =_{\mathrm{FV}(P)} s_1 = s =_{\mathrm{FV}(P)} s'$ and hence $s'_1 = s'$. We will show that $s'_{i+1} = [s_{i+1}, s'_i, \mathrm{FV}(P_1)]$ for $i = 1,\ldots,m-1$.

For that we will show that $s'_{i+1} =_{\mathrm{FV}(P_1)^c} s'_i$. By (1), $s_{i+1} =_{\mathrm{Mod}_1(P_1)^c} s_i$ and hence $s_{i+1} =_{\mathrm{FV}(P_1)^c} s_i$. For all $y \in \mathrm{FV}(P) - \mathrm{FV}(P_1)$, $s'_{i+1}(y) = s_{i+1}(y) = s_i(y) = s'_i(y)$. For all $y \notin \mathrm{FV}(P)$, $s'_{i+1}(y) = s'(y) = s'_i(y)$. Hence we have $s'_{i+1} =_{\mathrm{FV}(P_1)^c} s'_i$.

Since $s'_i =_{\mathrm{FV}(P)} s_i$, we have $s'_{i+1} =_{\mathrm{FV}(P_1)} s_{i+1}$ and then $s'_{i+1} = [s_{i+1}, s'_i, \mathrm{FV}(P_1)]$.

Then by (2), we have $[\![P_1]\!]^-((s'_i,h_i)) \ni (s'_{i+1},h_{i+1})$ for all $i = 1,\ldots,m-1$. Since $s'_m =_{\mathrm{FV}(P)} s_m$, we also have $[\![b]\!]_{s'_m} = \text{True}$, $s'_m =_{\mathrm{FV}(P_1)} s_m$ and then by induction hypothesis $[\![P_1]\!]^-((s'_m,h_m)) \ni \text{abort}$. Now we have $(s',h) = (s'_1,h_1)$, $[\![P_1]\!]^-((s'_i,h_i)) \ni (s'_{i+1},h_{i+1})$ and $[\![b]\!]_{s'_i} = \text{True}$ for all $i = 1,\ldots,m-1$, $[\![P_1]\!]^-((s'_m,h_m)) \ni \text{abort}$ and $[\![b]\!]_{s'_m} = \text{True}$. Therefore, by Proposition 4.2.4, $[\![\text{while }(b)\text{ do }(P)]\!]^-((s',h)) \ni \text{abort}$.

Case 3.2. $[\![b]\!]_s = \text{False}$. Then $[\![b]\!]_{s'} = \text{False}$. Assume $[\![\text{while }(b)\text{ do }(P)]\!]^-((s,h)) \ni \text{abort}$. By definition it is false.

Case 4. $P_1; P_2$.

Assume $s =_{\mathrm{FV}(P_1;P_2)} s'$ and $[\![P_1;P_2]\!]((s,h)) \ni \text{abort}$. By definition, $\bigcup\{\, [\![P_2]\!]^-(r) \mid r \in [\![P_1]\!]^-((s,h)) \,\} \ni \text{abort}$. Then either $[\![P_1]\!]^-((s,h)) \ni \text{abort}$ or $[\![P_1]\!]^-((s,h)) \ni (s_1,h_1)$ and $[\![P_2]\!]^-((s_1,h_1)) \ni \text{abort}$ for some s_1, h_1. Assume $[\![P_1]\!]^-((s,h)) \ni \text{abort}$. Since $\mathrm{FV}(P_1) \subseteq \mathrm{FV}(P_1;P_2)$, we have $s =_{\mathrm{FV}(P_1)} s'$. By induction hypothesis, $[\![P_1]\!]^-((s',h)) \ni \text{abort}$. Since $[\![P_2]\!]^-(\text{abort}) \ni \text{abort}$, we have $[\![P_1;P_2]\!]^-((s',h)) \ni \text{abort}$.

Now assume, $[\![P_1]\!]^-((s,h)) \ni (s_1,h_1)$ and $[\![P_2]\!]^-((s_1,h_1)) \ni \text{abort}$. We have $s =_{\mathrm{FV}(P_1)} s'$. Let $s'_1 = [s_1, s', \mathrm{FV}(P_1)]$. Then by induction hypothesis (2), $[\![P_1]\!]^-((s',h_1)) \ni (s'_1,h_1)$.

By (1), $s_1 =_{\mathrm{FV}(P_1)^c} s$ and $s_1' =_{\mathrm{FV}(P_1)^c} s'$. Then for all $y \in \mathrm{FV}(P_2) \cap \mathrm{FV}(P_1)$, $s_1(y) = s_1'(y)$. Since $\mathrm{FV}(P_2) - \mathrm{FV}(P_1) \subseteq \mathrm{Mod}_1(P_1)^c$, for all $y \in \mathrm{FV}(P_2) - \mathrm{FV}(P_1)$, we have $s_1(y) = s(y) = s'(y) = s_1'(y)$. Then $s_1 =_{\mathrm{FV}(P_2)} s_1'$. By induction hypothesis, $[\![P_2]\!]^-((s_1', h_1)) \ni \mathrm{abort}$. Then, $[\![P_1; P_2]\!]^-((s_1', h_1)) \ni \mathrm{abort}$.

Case 5. skip.

Its proof is immediate.

Case 6. $x := \mathrm{cons}(e_1, e_2)$.

Assume $s =_{\mathrm{FV}(x:=\mathrm{cons}(e_1,e_2))} s'$ and $[\![x := \mathrm{cons}(e_1, e_2)]\!]^-((s, h)) \ni \mathrm{abort}$. But by definition, $[\![x := \mathrm{cons}(e_1, e_2)]\!]^-((s, h)) \not\ni \mathrm{abort}$. Then $[\![x := \mathrm{cons}(e_1, e_2)]\!]^-((s, h)) \ni$ abort implies $[\![x := \mathrm{cons}(e_1, e_2)]\!]^-((s', h)) \ni \mathrm{abort}$.

Case 7. $x := [e]$.

Assume $s =_{\mathrm{FV}(x:=[e])} s'$ and $[\![x := [e]]\!]^-((s, h)) \ni \mathrm{abort}$. Then by definition, $[\![e]\!]_s \notin \mathrm{Dom}(h)$. Since $s =_{\mathrm{FV}(x:=[e])} s'$, we have $[\![e]\!]_s = [\![e]\!]_{s'}$. Then $[\![e]\!]_{s'} \notin \mathrm{Dom}(h)$. Therefore, $[\![x := [e]]\!]^-((s', h)) \ni \mathrm{abort}$.

Case 8. $[e_1] := e_2$.

Assume $s =_{\mathrm{FV}([e_1]:=e_2)} s'$ and $[\![[e_1] := e_2]\!]^-((s, h)) \ni \mathrm{abort}$. Then by definition, $[\![e_1]\!]_s \notin \mathrm{Dom}(h)$. Since $s =_{\mathrm{FV}([e_1]:=e_2)} s'$, we have $[\![e_1]\!]_s = [\![e_1]\!]_{s'}$. Then $[\![e_1]\!]_{s'} \notin \mathrm{Dom}(h)$. Therefore, $[\![[e_1] := e_2]\!]^-((s', h)) \ni \mathrm{abort}$.

Case 9. dispose(e).

Assume $s =_{\mathrm{FV}(\mathrm{dispose}(e))} s'$ and $[\![\mathrm{dispose}(e)]\!]^-((s, h)) \ni \mathrm{abort}$. Then by definition, $[\![e]\!]_s \notin \mathrm{Dom}(h)$. Since $s =_{\mathrm{FV}(\mathrm{dispose}(e))} s'$, we have $[\![e]\!]_s = [\![e]\!]_{s'}$. Then $[\![e]\!]_{s'} \notin \mathrm{Dom}(h)$. Therefore, $[\![\mathrm{dispose}(e)]\!]^-((s', h)) \ni \mathrm{abort}$. □

Lemma 4.2.16 *Suppose* $P \in \mathcal{L}$.

(1) If $s =_{EFV(P)} s'$ *and* $[\![P]\!]((s, h)) \ni$ *abort, then* $[\![P]\!]((s', h)) \ni$ *abort.*

(2) If $s =_{EFV(P)} s'$ *and* $[\![P]\!]((s, h)) \ni (s_1, h_1)$ *and* $s_1' = [s_1, s', EFV(P)]$ *then* $[\![P]\!]((s', h)) \ni (s_1', h_1)$.

(3) *If* $[\![P]\!]((s,h)) \ni (s_1,h_1)$ *then* $s_1 =_{Mod(P)^c} s$.

Proof. (1) Assume $s =_{\mathrm{EFV}(P)} s'$ and $[\![P]\!]((s,h)) \ni$ abort. Then by definition, $[\![P^{(k)}]\!]^-((s,h)) \ni$ abort for some k. Since we have $\mathrm{FV}(P^{(k)}) \subseteq \mathrm{EFV}(P)$ by Proposition 4.1.14 (1), we have $s =_{\mathrm{FV}(P^{(k)})} s'$. By Lemma 4.2.15 (3), we have $[\![P^{(k)}]\!]^-((s',h)) \ni$ abort. Then by definition, $[\![P]\!]((s',h)) \ni$ abort.

(2) Assume $s =_{\mathrm{EFV}(P)} s'$, $[\![P]\!]((s,h)) \ni (s_1,h_1)$ and $s'_1 = [s_1, s', \mathrm{EFV}(P)]$. Then by definition, $[\![P^{(k_1)}]\!]((s,h)) \ni (s_1,h_1)$ for some k_1. By definition, $\mathrm{EFV}(P) = \mathrm{FV}(P^{(n_{proc})})$ where n_{proc} is the number of our procedure names. Let $k = max(k_1, n_{proc})$. Then by Lemma 4.2.8, $[\![P^{(k)}]\!]((s,h)) \ni (s_1,h_1)$. Since $\mathrm{FV}(P^{(n_{proc})}) = \mathrm{FV}(P^{(k)})$ by Proposition 4.1.14 (1), we have $\mathrm{FV}(P^{(k)}) = \mathrm{EFV}(P)$. Then $s'_1 = [s_1, s', \mathrm{FV}(P^{(k)})]$. Then by Lemma 4.2.15 (2), $[\![P^{(k)}]\!]^-((s',h)) \ni (s'_1,h_1)$. Therefore, $[\![P]\!]((s',h)) \ni (s'_1,h_1)$.

(3) We can show the claim in a similar way to (1). □

4.3 Logical System

This section defines our logical system. It is the extension of Reynolds' system presented in [18] for mutual recursive procedure call.

We will write $A[x := e]$ for the formula obtained from A by replacing x by e. We will write the formula $e \mapsto e_1, e_2$ to denote $(e \mapsto e_1) * (e + 1 \mapsto e_2)$.

Definition 4.3.1 *Our logical system consists of the following inference rules. As mentioned in the previous section, we will use Γ for a set of asserted programs. A judgment is defined as $\Gamma \vdash \{A\}P\{B\}$.*

Skip

$$\frac{}{\vdash \{A\}skip\{A\}}$$

Identity

$$\frac{}{\{\{A\}P\{B\}\} \vdash \{A\}P\{B\}}$$

Assignment

$$\frac{}{\vdash \{A[x := e]\}x := e\{A\}}$$

If

$$\frac{\Gamma \vdash \{A \wedge b\}P_1\{B\} \quad \Gamma \vdash \{A \wedge \neg b\}P_2\{B\}}{\Gamma \vdash \{A\}if\ (b)\ then\ (P_1)\ else\ (P_2)\{B\}}$$

While

$$\frac{\Gamma \vdash \{A \wedge b\}P\{A\}}{\Gamma \vdash \{A\}while\ (b)\ do\ (P)\{A \wedge \neg b\}}$$

Composition

$$\frac{\Gamma \vdash \{A\}P_1\{C\} \quad \Gamma \vdash \{C\}P_2\{B\}}{\Gamma \vdash \{A\}P_1; P_2\{B\}}$$

Conseq

$$\frac{\Gamma \vdash \{A_1\}P\{B_1\}}{\Gamma \vdash \{A\}P\{B\}} \quad (A \to A_1, B_1 \to B)$$

Cons

$$\frac{}{\vdash \{\forall x'((x' \mapsto e_1, e_2) -\!\!* A[x := x'])\}x := cons(e_1, e_2)\{A\}} \ (x' \notin FV(e_1, e_2, A))$$

Lookup

$$\frac{}{\vdash \{\exists x'(e \mapsto x' * (e \mapsto x' -\!\!* A[x := x']))\}x := [e]\{A\}} \ (x' \notin FV(e, A))$$

Mutation

$$\frac{}{\vdash \{(\exists x(e_1 \mapsto x)) * (e_1 \mapsto e_2 -\!\!* A)\}[e_1] := e_2\{A\}} \ (x \notin FV(e_1))$$

Dispose

$$\frac{}{\vdash \{(\exists x(e \mapsto x)) * A\}dispose(e)\{A\}} \ (x \notin FV(e))$$

Recursion

$$\frac{\begin{array}{c}\Gamma \cup \{\{A_i\}R_i\{B_i\} | i = 1, \ldots, n_{proc}\} \vdash \{A_1\}Q_1\{B_1\} \\ \vdots \\ \Gamma \cup \{\{A_i\}R_i\{B_i\} | i = 1, \ldots, n_{proc}\} \vdash \{A_{n_{proc}}\}Q_{n_{proc}}\{B_{n_{proc}}\}\end{array}}{\Gamma \vdash \{A_j\}R_j\{B_j\}} \quad (1 \leq j \leq n_{proc})$$

Inv-Conj

$$\frac{\Gamma \vdash \{A\}P\{C\}}{\Gamma \vdash \{A \wedge B\}P\{C \wedge B\}} \ (FV(B) \cap Mod(P) = \emptyset, B \text{ is pure})$$

Exists

$$\frac{\Gamma \vdash \{A\}P\{B\}}{\Gamma \vdash \{\exists x.A\}P\{B\}} \ (x \notin FV(B) \cup EFV(P))$$

Cut

$$\frac{\Gamma \vdash \{A_1\}P_1\{B_1\} \quad \Gamma \cup \{\{A_1\}P_1\{B_1\}\} \vdash \{A\}P\{B\}}{\Gamma \vdash \{A\}P\{B\}}$$

Weakening

$$\frac{\Gamma \vdash \{A\}P\{B\}}{\Gamma \cup \Gamma' \vdash \{A\}P\{B\}}$$

We say $\{A\}P\{B\}$ is provable and we write $\vdash \{A\}P\{B\}$, when $\vdash \{A\}P\{B\}$ can be derived by these inference rules.

We explain inference rules by the following simple example.

4.3.1 Example

Let A be the abbreviation of $\forall y(y \geq x \wedge y < z \leftrightarrow \exists w(y \mapsto w) * \text{True})$. So A asserts about the heap which has the domain $\{x+0, x+1, \ldots, x+k\}$ where $k+1=z$. Suppose we have the procedure declaration

$$\text{Procedure } R_1(\text{if } (x < z) \text{ then } (\text{dispose}(x); x := x+1; R_1) \text{ else } (\text{skip}))$$

We will show that $\vdash \{A\}R_1\{\text{emp}\}$ is provable in our system. This means we have a heap with some consecutive allocation of cells and the program R_1 deallocates them all.

Let Γ be $\{\{A\}R_1\{\text{emp}\}\}$. The axiom (Assignment) and the rule (Weakening) gives

$$\Gamma \vdash \{A[x := x+1]\}x := x+1\{A\}.$$

The axiom (Identity) gives

$$\Gamma \vdash \{A\}R_1\{\mathrm{emp}\}.$$

By the inference rule (Composition), we have

$$\Gamma \vdash \{A[x := x + 1]\}x := x + 1; R_1\{\mathrm{emp}\}.$$

The axiom (Dispose) and the rule (Weakening) gives

$$\Gamma \vdash \{(\exists y(x \mapsto y)) * A[x := x + 1]\}\mathrm{dispose}(x)\{A[x := x + 1]\}.$$

Then by the inference rule (Composition), we have

$$\Gamma \vdash \{(\exists y(x \mapsto y)) * A[x := x + 1]\}\mathrm{dispose}(x); x := x + 1; R_1\{\mathrm{emp}\}.$$

The axiom (Skip) and the rule (Weakening) gives

$$\Gamma \vdash \{A \wedge \neg(x < z)\}\mathrm{skip}\{A \wedge \neg(x < z)\}.$$

Indeed $A \wedge \neg(x < z)$ is true only for the empty heap. Now we have $A \wedge x < z \to (\exists y(x \mapsto y)) * A[x := x + 1]$ and $A \wedge \neg(x < z) \to \mathrm{emp}$. Then by the inference rule (Conseq), we have

$$\Gamma \vdash \{A \wedge x < z\}\mathrm{dispose}(x); x := x + 1; R_1\{\mathrm{emp}\}$$

and

$$\Gamma \vdash \{A \wedge \neg(x < z)\}\mathrm{skip}\{\mathrm{emp}\}.$$

By applying the inference rule (if), we get

$$\Gamma \vdash \{A\}\mathrm{if}\,(x < z)\ \mathrm{then}\ (\mathrm{dispose}(x); x := x + 1; R_1)\ \mathrm{else}\ (\mathrm{skip})\{\mathrm{emp}\}.$$

Finally, by the inference rule (Recursion), we have $\vdash \{A\}R_1\{\mathrm{emp}\}$ is provable.

4.3.2 New Rules

We have two new rules, (Inv-Conj) and (Exists), which appear neither in [3] nor in [18].

We have found that the axiom (Axiom 9: Invariance Axiom) in [3] is not sound for our system. The definition of the axiom will be given in the next chapter. The asserted program $\{x = 0\}[y] := 0\{x = 0\}$ is a counter example since it is not true although it is provable by the axiom. It causes abort for the state $(s, h) = (s'[x := 0, y := 1], \emptyset)$ though $[\![x = 0]\!]_{(s,h)}$ is true. Another counter example is $\{\mathrm{emp}\}x := \mathrm{cons}(0, 0)\{\mathrm{emp}\}$. Although it does not abort, it is not true. So, we have introduced the rule (Inv-Conj). It is a variant of the combination of the axiom (Axiom 9: Invariance Axiom) and the rule (Rule 12: Conjunction Rule) of [3]. That is, an arbitrary assertion, also called an invariant [23], can be conjuncted to the precondition and postcondition of an asserted program if none of its free variables can be modified by the program. Note that it is the frame rule [18] restricted to pure formulas.

The rule (Exists) is analogous to the rule existential introduction of propositional calculus. Rule 15: Elimination Rule in [3] is similar to this inference rule.

5
Soundness and Completeness

5.1 Soundness

In this section we will prove soundness of our system. That means we will show that if a judgment $\Gamma \vdash \{A\}P\{B\}$ is derivable in our system then $\Gamma \vdash \{A\}P\{B\}$ is also true.

We say an unfolded program is correct when eac level of the unfolding of the program is correct. For a given specification, correctness of a program is the same as that of its unfolded transformation.

It is straightforward to construct a logical system by collecting axioms and inference rules from both Hoare's logic for recursive procedure and separation logic to verify pointer programs with recursive procedures. But this system is unsound. It is because the invariance axiom is not sound in separation logic. At the end of this section, we will give an unsound example.

Proposition 5.1.1 *$\{A\}P\{B\}$ is true if and only if for all k, $\{A\}P^{(k)}\{B\}$ is true.*

Proof. First we will show from the left-hand side to the right-hand side. Assume $\{A\}P\{B\}$ is true. Assume $[\![A]\!]_r = \text{True}$. Then by definition, $[\![P]\!](r) \not\ni \text{abort}$. Then by definition, $\bigcup_{k=0}^{\infty}([\![P^{(k)}]\!]^-(r)) \not\ni \text{abort}$. Then for all k, $[\![P^{(k)}]\!]^-(r) \not\ni \text{abort}$. Fix k. Assume $[\![P^{(k)}]\!]^-(r) \ni r'$. Then we have $\bigcup_{k=0}^{\infty}([\![P^{(k)}]\!]^-(r)) \ni r'$. Then again by definition, $[\![P]\!](r) \ni r'$. Then $[\![B]\!]_{r'} = \text{True}$. Then $\{A\}P^{(k)}\{B\}$ is true for all k.

Next we will show the right-hand side to the left-hand side. Assume $\{A\}P^{(k)}\{B\}$ is true for all k. Assume $[\![A]\!]_r = \text{True}$. Fix k. Then by definition, $[\![P^{(k)}]\!](r) \not\ni \text{abort}$. Then $[\![P^{(k)}]\!]^-(r) \not\ni \text{abort}$. Then $\bigcup_{k=0}^{\infty}([\![P^{(k)}]\!]^-(r)) \not\ni \text{abort}$. Then by definition, $[\![P]\!](r) \not\ni \text{abort}$. Assume $[\![P]\!](r) \ni r'$. Then by definition, $\bigcup_{k=0}^{\infty}([\![P^{(k)}]\!]^-(r)) \ni r'$. Then we have $[\![P^{(k)}]\!]^-(r) \ni r'$ for some k. Then by definition, $[\![P^{(k)}]\!](r) \ni r'$. Then $[\![B]\!]_{r'} = \text{True}$. Then $\{A\}P\{B\}$ is true. □

Definition 5.1.2 *For a set Γ of asserted programs, $\Gamma^{(k)}$ is defined by $\{\, \{A_i\}P_i\{B_i\} \mid 1 \le i \le n \,\}^{(k)} = \{\, \{A_i\}P_i^{(k)}\{B_i\} \mid 1 \le i \le n \,\}$.*

If the truth of an assertion depends only on a store, it is not altered in the presence of any heap. This easy lemma will be necessary for some formal discussions later.

Lemma 5.1.3 *$[\![A]\!]_s = [\![A]\!]_{(s,h)}$ for a pure formula A where the left-hand side is the semantics for the base language and the right-hand side is the semantics for the assertion language.*

Proof. This is proved by induction on A. If A is $B \wedge C$, we have $[\![A]\!]_s = ([\![B]\!]_s$ and $[\![C]\!]_s)$ and $[\![A]\!]_{(s,h)} = ([\![B]\!]_{(s,h)}$ and $[\![C]\!]_{(s,h)})$, and both sides are the same, since by the induction hypothesis we have $[\![B]\!]_s = [\![B]\!]_{(s,h)}$ and $[\![C]\!]_s = [\![C]\!]_{(s,h)}$. Other cases are similarly proved. $\square$

The following lemma is important for the soundness of the inference rule (Recursion).

Lemma 5.1.4 *If $\{ \{A_i\}R_i^{(k)}\{B_i\} \mid 1 \le i \le n_{proc} \} \vdash \{A_i\}Q_i^{(k)}\{B_i\}$ is true for all i and k then $\vdash \{A_i\}R_i^{(k)}\{B_i\}$ true for all i.*

Proof. Assume $\{ \{A_i\}R_i^{(k)}\{B_i\} \mid 1 \le i \le n_{proc} \} \vdash \{A_i\}Q_i^{(k)}\{B_i\}$ is true for all i and k. We will show $\{A_i\}R_i^{(k)}\{B_i\}$ true for all i by induction on k.

Case 1. $k = 0$.

By Proposition 4.1.10 (2) $R_i^{(0)} = \Omega$. Then $\{A_i\}\Omega\{B_i\}$ is true.

Case 2. $k = k' + 1$.

We assume that $\{ \{A_i\}R_i^{(k)}\{B_i\} \mid 1 \le i \le n_{proc} \} \vdash \{A_i\}Q_i^{(k)}\{B_i\}$ is true for all k. By induction hypothesis, $\{A_i\}R_i^{(k')}\{B_i\}$ is true for all i. By the assumption for k', we have $\{ \{A_i\}R_i^{k'}\{B_i\} \mid 1 \le i \le n_{proc} \} \vdash \{A_i\}Q_i^{k'}\{B_i\}$ true. Hence $\{A_i\}Q_i^{(k')}\{B_i\}$ is true. By Proposition 4.1.10 (3), $R_i^{(k'+1)} = Q_i^{(k')}$. Hence $\{A_i\}R_i^{(k)}\{B_i\}$ is true for all i. $\square$

In the following lemma we consider an assertion and a program such that the assertion has no free variable that can be modified by the program. In such a case, the truth of the assertion is preserved even after the execution of the program.

Lemma 5.1.5 *If A is pure, P doesn't contain any procedure names, $[\![A]\!]_{(s,h)} = True$, $FV(A) \cap Mod_1(P) = \emptyset$ and $[\![P]\!]^-((s,h)) \ni (s',h')$ then $[\![A]\!]_{(s',h')} = True$.*

Proof. Assume A is *pure*, P doesn't contain any procedure names, $[\![A]\!]_{(s,h)} = \text{True}$,

$\mathrm{FV}(A) \cap \mathrm{Mod}_1(P) = \emptyset$ and $[\![P]\!]^-((s,h)) \ni (s',h')$. By Lemma 4.2.15 (1), $s =_{\mathrm{Mod}(P)^c} s'$. Since $\mathrm{FV}(A) \subseteq \mathrm{Mod}_1(P)^c$, we have $s =_{\mathrm{FV}(A)} s'$. By Lemma 5.1.3, $[\![A]\!]_s$=True. Hence $[\![A]\!]_{s'}$=True. By Lemma 5.1.3, $[\![A]\!]_{(s',h')} = \mathrm{True}$. □

The following lemma will be necessary to show the soundness of the inference rule (Inv-Conj).

Lemma 5.1.6 *If P doesn't contain any procedure names, B is pure, $Mod_1(P) \cap FV(B) = \emptyset$ and $\{A\}P\{C\}$ is true, then $\{A \wedge B\}P\{C \wedge B\}$ is true.*

Proof. Assume B is *pure*, $\mathrm{Mod}_1(P) \cap \mathrm{FV}(B) = \emptyset$ and $\{A\}P\{C\}$ is true. We have to prove $\{A \wedge B\}P\{C \wedge B\}$ is true.

Assume $[\![A \wedge B]\!]_{(s,h)} = \mathrm{True}$. Then $[\![A]\!]_{(s,h)} = \mathrm{True}$ and $[\![B]\!]_{(s,h)} = \mathrm{True}$. Then $[\![P]\!]((s,h)) \not\ni \mathrm{abort}$. Assume $[\![P]\!]((s,h)) \ni (s',h')$. Then $[\![C]\!]_{(s',h')} = \mathrm{True}$. By Lemma 5.1.5, $[\![B]\!]_{(s',h')} = \mathrm{True}$. Then $[\![B \wedge C]\!]_{(s',h')} = \mathrm{True}$. Hence, $\{A \wedge B\}P\{C \wedge B\}$ is true. □

Lemma 5.1.7 *If $\Gamma \vdash \{A\}P\{B\}$ is provable then $\Gamma^{(k)} \vdash \{A\}P^{(k)}\{B\}$ is true for all k.*

Proof. It is proved by induction on the proof. We consider cases according to the last rule.

Case 1. (Skip).

Its proof is immediate.

Case 2. (Identity).

Assume $\Gamma, \{A\}P\{B\} \vdash \{A\}P\{B\}$ is provable. Then $\Gamma^{(k)}, \{A\}P^{(k)}\{B\} \vdash \{A\}P^{(k)}\{B\}$ is true for all k.

Case 3. (Assignment).

Assume $\Gamma \vdash \{A[x := e]\}x := e\{A\}$ is provable. Assume $[\![A[x := e]]\!]_{(s,h)} = \mathrm{True}$. Let n be $[\![e]\!]_s$. By the definition we have $[\![x := e]\!]((s,h)) = \{(s_1,h)\}$ where $s_1 = s[x := n]$. Since $[\![A[x := e]]\!]_{(s,h)} = [\![A]\!]_{(s_1,h)}$, we have $[\![A]\!]_{(s_1,h)} = \mathrm{True}$. Hence $\{A[x := e]\}x := e\{A\}$ is true. Then by definition, $\Gamma^{(k)} \vdash \{A[x := e]\}(x := e)^{(k)}\{A\}$

is true.

Case 4. (If).

We have the asserted program $\Gamma \vdash \{A \wedge b\}P_1\{B\}$ and $\Gamma \vdash \{A \wedge \neg b\}P_2\{B\}$ which are provable. By induction hypothesis, $\Gamma^{(k)} \vdash \{A \wedge b\}P_1^{(k)}\{B\}$ and $\Gamma^{(k)} \vdash \{A \wedge \neg b\}P_2^{(k)}\{B\}$ are true.

Now we will show that $\Gamma^{(k)} \vdash \{A\}(\text{if}\,(b)\text{ then }(P_1)\text{ else }(P_2))^{(k)}\{B\}$ is true for all k. Assume $\Gamma^{(k)}$ is true. Then $\{A \wedge b\}P_1^{(k)}\{B\}$ and $\{A \wedge \neg b\}P_2^{(k)}\{B\}$ are true.

Assume $[\![A]\!]_{(s,h)}$ is true.

Case 4.1. $[\![b]\!]_s$ is true. By Lemma 5.1.3, we have $[\![b]\!]_{(s,h)}$ is true. By definition, $[\![A \wedge b]\!]_{(s,h)}$ is true. Assume $[\![(\text{if}\,(b)\text{ then }(P_1)\text{ else }(P_2))^{(k)}]\!]((s,h)) = [\![P_1^{(k)}]\!]((s,h)) \ni r$. Then $r \neq \text{abort}$ and $[\![B]\!]_r$ is true.

Case 4.2. $[\![b]\!]_s$ is false. Then the fact that $r \neq$ abort and $[\![B]\!]_r$ is true is similarly proved to Case 1.

Then $\{A\}(\text{if}\,(b)\text{ then }(P_1)\text{ else }(P_2))^{(k)}\{B\}$ is true. Then by definition $\Gamma^{(k)} \vdash \{A\}(\text{if }b\text{ then }P_1\text{ else }P_2)^{(k)}\{B\}$ is true for all k.

Case 5. (While).

B is in the form $A \wedge \neg b$ and by (while) rule we have the asserted program $\Gamma \vdash \{A \wedge b\}P\{A\}$ which is provable. By induction hypothesis, $\Gamma^{(k)} \vdash \{A \wedge b\}P^{(k)}\{A\}$ is true for all k.

Now we will show that $\Gamma^{(k)} \vdash \{A\}(\text{while}\,(b)\text{ do }(P))^{(k)}\{\neg b \wedge A\}$ is true. Assume $\Gamma^{(k)}$ is true. Then $\{A \wedge b\}P^{(k)}\{A\}$ is true.

Let define the function $F : \text{States} \cup \{\text{abort}\} \to p(\text{States} \cup \{\text{abort}\})$ by

$$F(\text{abort}) = \{\text{abort}\},$$
$$F((s,h)) = \{(s_0,h_0) \mid [\![A \wedge \neg b]\!]_{(s_0,h_0)} = \text{True}\} \cup \{r \in \text{States} \mid [\![A]\!]_{(s,h)} = \text{False}\}.$$

We will show that F satisfies the inequations obtained from the equations for

$[\![(\text{while }(b)\text{ do }(P))^{(k)}]\!]$ by replacing $=$ by $\supseteq$. That is, we will show

$$\begin{aligned}
&F(\text{abort}) \supseteq \{\text{abort}\},\\
&F((s,h)) \supseteq \{(s,h)\} \text{ if } [\![b]\!]_s = \text{False},\\
&F((s,h)) \supseteq \bigcup\{\, F(r) \mid r \in [\![P^{(k)}]\!]((s,h)) \,\} \text{ if } [\![b]\!]_s = \text{True}.
\end{aligned}$$

The first inequation immediately holds by the definition of F. We will show the second inequation. Assume $[\![b]\!]_s = \text{False}$. By Lemma 5.1.3, we have $[\![b]\!]_{(s,h)} = \text{False}$. If $[\![A]\!]_{(s,h)} = \text{False}$, then the inequation holds by the definition of F. If $[\![A]\!]_{(s,h)} = \text{True}$, then $[\![A \wedge \neg b]\!]_{(s,h)} = \text{True}$, and the inequation holds by the definition of F.

We will show the last inequation. If $[\![A]\!]_{(s,h)} = \text{False}$, then it holds by the definition of F. Assume $[\![A]\!]_{(s,h)} = \text{True}$, $[\![b]\!]_s = \text{True}$, and r' is in the right-hand side. We will show that $r' \in F((s,h))$. We have some r_1 such that $[\![P^{(k)}]\!]((s,h)) \ni r_1$ and $F(r_1) \ni r'$. By Lemma 5.1.3, we have $[\![b]\!]_{(s,h)} = \text{True}$. Hence $[\![A \wedge b]\!]_{(s,h)} = \text{True}$. Since $\{A \wedge b\}P^{(k)}\{A\}$ is true we have $r_1 \neq \text{abort}$ and $[\![A]\!]_{r_1} = \text{True}$. Then we have $r' \neq \text{abort}$ and $[\![A \wedge \neg b]\!]_{r'} = \text{True}$. Hence $r' \in F((s,h))$.

We have shown that F satisfies the inequations. By the least fixed point theorem [22], we have $F \supseteq [\![(\text{while }(b)\text{ do }(P))^{(k)}]\!]$. If $[\![A]\!]_{(s,h)} = \text{True}$ and $r' \in [\![(\text{while }(b)\text{ do }(P))^{(k)}]\!]((s,h))$, then $r' \in F((s,h))$, and we have $r' \neq \text{abort}$ and $[\![A \wedge \neg b]\!]_{r'} = \text{True}$. Therefore $\Gamma^{(k)} \vdash \{A\}(\text{while }(b)\text{ do }(P))^{(k)}\{A \wedge \neg b\}$ is true.

Case 6. (Composition).

We have the asserted program $\Gamma \vdash \{A\}P_1\{C\}$ and $\Gamma \vdash \{C\}P_2\{B\}$ which are provable for some C. By induction hypothesis, $\Gamma^{(k)} \vdash \{A\}P_1^{(k)}\{C\}$ and $\Gamma \vdash \{C\}P_2^{(k)}\{B\}$ are true for all k.

Now we will show that $\Gamma^{(k)} \vdash \{A\}(P_1;P_2)^{(k)}\{B\}$ is true for all k. Assume $\Gamma^{(k)}$ is true. Then $\{A\}P_1^{(k)}\{C\}$ and $\{C\}P_2^{(k)}\{B\}$ are true.

Assume $[\![A]\!]_{(s,h)} = \text{True}$ and $[\![(P_1;P_2)^{(k)}]\!]((s,h)) \ni r$. We will show $r \neq \text{abort}$ and $[\![B]\!]_r = \text{True}$. We have r_1 such that $[\![P_1^{(k)}]\!]((s,h)) \ni r_1$ and $[\![P_2^{(k)}]\!](r_1) \ni r$. Hence we have $r_1 \neq \text{abort}$ and $[\![C]\!]_{r_1} = \text{True}$. Since $[\![P_2^{(k)}]\!](r_1) \ni r$, we have $r \neq \text{abort}$ and $[\![B]\!]_r = \text{True}$. Hence $\{A\}(P_1;P_2)^{(k)}\{B\}$ is true. Then $\Gamma^{(k)} \vdash \{A\}(P_1;P_2)^{(k)}\{B\}$ is true for all k.

Case 7. (Conseq).

We have the asserted program $\Gamma \vdash \{A'\}P\{B'\}$ which is provable where $A \to A'$ and $B' \to B$. By induction hypothesis for the assumption, $\Gamma^{(k)} \vdash \{A'\}P^{(k)}\{B'\}$ is true for all k.

Now we will show that $\Gamma^{(k)} \vdash \{A\}P^{(k)}\{B\}$ is true for all k. Assume $\Gamma^{(k)}$ is true. Then $\{A'\}P^{(k)}\{B'\}$ is true.

Assume $[\![A]\!]_{(s,h)} = \text{True}$ and $[\![P^{(k)}]\!]((s,h)) \ni r$. We will show $r \neq$ abort and $[\![B]\!]_r = \text{True}$. By the side condition $[\![A \to A_1]\!]_{(s,h)} = \text{True}$, we have $[\![A_1]\!]_{(s,h)} = \text{True}$. Hence $r \neq$ abort and $[\![B_1]\!]_r = \text{True}$. By the side condition $[\![B_1 \to B]\!]_r = \text{True}$, we have $[\![B]\!]_r = \text{True}$. Hence $\{A\}P^{(k)}\{B\}$ is true. Then $\Gamma^{(k)} \vdash \{A\}P^{(k)}\{B\}$ is true for all k.

Case 8. (Cons).

Assume $\Gamma \vdash \{\forall x'((x' \mapsto e_1, e_2) -\!\!* A[x := x'])\}x := \text{cons}(e_1, e_2)\{A\}$ is provable. Assume $[\![\forall x'(x' \mapsto e_1, e_2 -\!\!* A[x := x'])]\!]_{(s,h)} = \text{True}$ and $[\![x := \text{cons}(e_1, e_2)]\!]((s,h)) \ni r$. We will show $r \neq$ abort and $[\![A]\!]_r = \text{True}$. Let P be $x := \text{cons}(e_1, e_2)$, n_1 be $[\![e_1]\!]_s$, and n_2 be $[\![e_2]\!]_s$. By the definition of $[\![P]\!]$, $r \neq$ abort and $r = (s_1, h_1)$ where $s_1 = s[x := n]$ and $h_1 = h[n := n_1, n+1 := n_2]$ for some n such that $n > 0$ and $n, n+1 \notin \text{Dom}(h)$. By $[\![\forall x'(x' \mapsto e_1, e_2 -\!\!* A[x := x'])]\!]_{(s,h)} = \text{True}$, we have $[\![x' \mapsto e_1, e_2 -\!\!* A[x := x']]\!]_{(s',h)} = \text{True}$ where $s' = s[x' := n]$. Let $h_2 = \emptyset[n := n_1, n+1 := n_2]$. We have $h_1 = h + h_2$. Then $[\![x' \mapsto e_1, e_2]\!]_{(s',h_2)} = \text{True}$ since $x' \notin \text{FV}(e_1, e_2)$. Since $[\![x' \mapsto e_1, e_2 -\!\!* A[x := x']]\!]_{(s',h)} = \text{True}$, we have $[\![A[x := x']]\!]_{(s',h_1)} = \text{True}$. Hence $[\![A]\!]_{(s_1,h_1)} = \text{True}$, since $x' \notin \text{FV}(A)$. Now we have $\{\forall x'((x' \mapsto e_1, e_2) -\!\!* A[x := x'])\}x := \text{cons}(e_1, e_2)\{A\}$ is true. Then $\Gamma^{(k)} \vdash \{\forall x'((x' \mapsto e_1, e_2) -\!\!* A[x := x'])\}(x := \text{cons}(e_1, e_2))^{(k)}\{A\}$ is true for all k.

Case 9. (Lookup).

Assume $\Gamma \vdash \{\exists x'(e \mapsto x' * (e \mapsto x' -\!\!* A[x := x']))\}x := [e]\{A\}$ is provable. Assume $[\![\exists x'((e \mapsto x') * ((e \mapsto x') -\!\!* A[x := x']))]\!]_{(s,h)} = \text{True}$ and $[\![x := [e]]\!]((s,h)) \ni r$. We will show $r \neq$ abort and $[\![A]\!]_r = \text{True}$. Let n be

$[\![e]\!]_s$. We have some n_1 such that $[\![(e \mapsto x') * (e \mapsto x' \multimap A[x := x'])]\!]_{(s',h)} = \text{True}$ where $s' = s[x' := n_1]$. Hence we have h_1, h_2 such that $h_1 = \emptyset[n := n_1]$, $h = h_1 + h_2$ and $[\![e \mapsto x' \multimap A[x := x']]\!]_{(s',h_2)} = \text{True}$. By $[\![e \mapsto x']\!]_{(s',h_1)} = \text{True}$, $[\![A[x := x']]\!]_{(s',h)} = \text{True}$. Since $x' \notin \text{FV}(e)$, we have $[\![e]\!]_{s'} = [\![e]\!]_s = n$. Since $n \in \text{Dom}(h)$, we have $r \neq$ abort and $r = (s_1, h)$ where $s_1 = s[x := n_1]$. Since $[\![A[x := x']]\!]_{(s',h)} = [\![A]\!]_{(s_1,h)}$ by $x' \notin \text{FV}(A)$, we have $[\![A]\!]_r = \text{True}$. Then $\{\exists x'(e \mapsto x' * (e \mapsto x' \multimap A[x := x']))\}x := [e]\{A\}$ is true. Then we have $\{\Gamma^{(k)} \vdash \exists x'(e \mapsto x' * (e \mapsto x' \multimap A[x := x']))\}(x := [e])^{(k)}\{A\}$ is true for all k.

Case 10. (Mutation).

Assume $\Gamma \vdash \{(\exists x(e_1 \mapsto x)) * (e_1 \mapsto e_2 \multimap A)\}[e_1] := e_2\{A\}$ is provable. Assume $[\![(\exists x(e_1 \mapsto x)) * (e_1 \mapsto e_2 \multimap A)]\!]_{(s,h)} = \text{True}$ and $[\![[e_1] := e_2]\!]((s,h)) \ni r$. We will show $r \neq$ abort and $[\![A]\!]_r = \text{True}$. Let n be $[\![e_1]\!]_s$ and n_2 be $[\![e_2]\!]_s$. We have h_1, h_2 such that $[\![e_1 \mapsto e_2 \multimap A]\!]_{(s,h_2)}$ and $h = h_1 + h_2$ and $[\![\exists x(e_1 \mapsto x)]\!]_{(s,h_1)} = \text{True}$. Hence we have some n_1 such that $[\![e_1 \mapsto x]\!]_{(s',h_1)} = \text{True}$ where $s' = s[x := n_1]$. Since $x \notin \text{FV}(e_1)$, we have $[\![e_1]\!]_{s'} = [\![e_1]\!]_s = n$. We also have $\text{Dom}(h_1) = \{n\}$. Since $n \in \text{Dom}(h)$, $r \neq$ abort and $r = (s, h')$ where $h' = h[n := n_2]$. Let h'_1 be $\emptyset[n := n_2]$. Then $h' = h'_1 + h_2$ and $[\![e_1 \mapsto e_2]\!]_{(s,h'_1)} = \text{True}$. Since $[\![e_1 \mapsto e_2 \multimap A]\!]_{(s,h_2)} = \text{True}$, we have $[\![A]\!]_{(s,h')} = \text{True}$. Hence $\{(\exists x(e_1 \mapsto x)) * (e_1 \mapsto e_2 \multimap A)\}[e_1] := e_2\{A\}$ is true. Therefore $\Gamma^{(k)} \vdash \{(\exists x(e_1 \mapsto x)) * (e_1 \mapsto e_2 \multimap A)\}([e_1] := e_2)^{(k)}\{A\}$ is true for all k.

Case 11. (Dispose).

Assume $\Gamma \vdash \{(\exists x(e \mapsto x)) * A\}\text{dispose}(e)\{A\}$ is provable. Assume $[\![(\exists x(e \mapsto x)) * A]\!]_{(s,h)} = \text{True}$ and $[\![\text{dispose}(e)]\!]((s,h)) \ni r$. We will show $r \neq$ abort and $[\![A]\!]_r$. Let n be $[\![e]\!]_s$. Then we have h_1, h_2 such that $h = h_1 + h_2$, $[\![\exists x(e \mapsto x)]\!]_{(s,h_1)} = \text{True}$ and $[\![A]\!]_{(s,h_2)} = \text{True}$. Hence we have some n_1 such that $[\![e \mapsto x]\!]_{(s',h_1)} = \text{True}$ where $s' = s[x := n_1]$. Since $[\![e]\!]_{s'} = [\![e]\!]_s = n$ by $x \notin \text{FV}(e)$, $\text{Dom}(h_1) = \{n\}$ and $h_1(n) = n_1$. Since $n \in \text{Dom}(h)$, $r \neq$ abort and $r = (s, h'_2)$ where $h'_2 = h|_{\text{Dom}(h)-\{n\}}$. Hence $h_2 = h'_2$. Therefore $[\![A]\!]_r = \text{True}$. Hence $\{(\exists x(e \mapsto x)) * A\}\text{dispose}(e)\{A\}$ is true. Then $\Gamma^{(k)} \vdash \{(\exists x(e \mapsto x)) * A\}(\text{dispose}(e))^{(k)}\{A\}$ is true for all k.

Case 12. (Recursion).

We have the asserted program $\Gamma \cup \{ \{A_i\}R_i\{B_i\} \mid 1 \le i \le n_{proc} \} \vdash \{A_i\}Q_i\{B_i\}$ that are provable for all i. Fix i and k. By induction hypothesis, $\Gamma^{(k)} \cup \{ \{A_i\}R_i^{(k)}\{B_i\} \mid 1 \le i \le n_{proc} \} \vdash \{A_i\}Q_i^{(k)}\{B_i\}$ is true. Assume $\Gamma^{(k)}$ is true. Then $\{ \{A_i\}R_i^{(k)}\{B_i\} \mid 1 \le i \le n_{proc} \} \vdash \{A_i\}Q_i^{(k)}\{B_i\}$ is true.

By Lemma 5.1.4, $\vdash \{A_i\}R_i^{(k)}\{B_i\}$ is true. Then $\Gamma^{(k)} \vdash \{A_i\}R_i^{(k)}\{B_i\}$ is true.

Case 13. (Inv-Conj).

We have $\Gamma \vdash \{A\}P\{C\}$ is provable and $\mathrm{Mod}(P) \cap \mathrm{FV}(B) = \emptyset$. By induction hypothesis, $\Gamma^{(k)} \vdash \{A\}P^{(k)}\{C\}$ is true for all k.

Fix k. Now we will show that $\Gamma^{(k)} \vdash \{A \wedge B\}P^{(k)}\{C \wedge B\}$ is true. Assume $\Gamma^{(k)}$ is true. Then $\{A\}P^{(k)}\{C\}$ is true.

By definition, $\mathrm{Mod}(P) = \mathrm{Mod}_1(P^{(n_{proc})})$.

Since we have $\mathrm{Mod}_1(P^{(k)}) \subseteq \mathrm{Mod}_1(P^{(n_{proc})})$ hence $\mathrm{FV}(B) \cap \mathrm{Mod}_1(P^{(k)}) = \emptyset$. By Lemma 5.1.6, we have $\{A \wedge B\}P^{(k)}\{C \wedge B\}$ is true.

Case 14. (Exists).

We have the asserted program $\Gamma \vdash \{A\}P\{B\}$, which is provable and $x \notin \mathrm{FV}(B) \cup \mathrm{EFV}(P)$. By induction hypothesis, $\Gamma^{(k)} \vdash \{A\}P^{(k)}\{B\}$ is true for all k. Since $x \notin \mathrm{EFV}(P)$, by definition $x \notin \mathrm{FV}(P^{(n_{proc})})$. Since $\mathrm{FV}(P^{(k)}) \subseteq \mathrm{FV}(P^{(n_{proc})})$, we have $x \notin \mathrm{FV}(P^{(k)})$.

Fix k. Now we will show that $\Gamma^{(k)} \vdash \{\exists x A\}P^{(k)}\{B\}$ is true. Assume $\Gamma^{(k)}$ is true. Then $\{A\}P^{(k)}\{B\}$ is true.

Assume $[\![\exists x A]\!]_{(s,h)} = \mathrm{True}$ and $[\![P^{(k)}]\!]^-((s,h)) \ni r$. Then by definition $[\![A]\!]_{(s[x:=m],h)} = \mathrm{True}$ for some m. By definition $[\![P^{(k)}]\!]^-((s[x := m], h)) \not\ni \mathrm{abort}$. We have $s =_{\mathrm{FV}(P^{(k)})} s[x := m]$ since $x \notin \mathrm{FV}(P^{(k)})$. Then by Lemma 4.2.15(3), $r \neq \mathrm{abort}$. Assume $r = (s', h')$. Then by Lemma 4.2.15(1), we have $s' =_{\mathrm{Mod}_1(P^{(k)})^c} s$ and hence $s' =_{\mathrm{FV}(P^{(k)})^c} s$. Now let $s'_1 = [s', s[x := m], \mathrm{FV}(P^{(k)})]$. Then by Lemma 4.2.15(2), $[\![P^{(k)}]\!]^-((s[x := m], h)) \ni (s'_1, h')$. Then by definition, $[\![B]\!]_{(s'_1,h')} = \mathrm{True}$. We have $s'_1(y) = s'(y)$ for all $y \in \mathrm{FV}(P^{(k)})$. We also have $s'_1(y) = s(y) = s'(y)$ for all $y \neq x$ and

$y \notin \mathrm{FV}(P^{(k)})$. Hence $s'_1 =_{\{x\}^c} s'$. Since $x \notin \mathrm{FV}(B)$, we have $[\![B]\!]_{(s',h')} = \mathrm{True}$. Then by definition $\{\exists x A\}P^{(k)}\{B\}$ is true. Therefore, $\Gamma^{(k)} \vdash \{\exists x A\}P^{(k)}\{B\}$ is true. □

We present the soundness theorem. It is one of the important theorems in our paper.

Theorem 5.1.8 (Soundness) *If $\Gamma \vdash \{A\}P\{B\}$ is provable, $\Gamma \vdash \{A\}P\{B\}$ is true.*

Proof. Assume $\Gamma \vdash \{A\}P\{B\}$ is provable. By Lemma 5.1.7, $\Gamma^{(k)} \vdash \{A\}P^{(k)}\{B\}$ true for all k. By Proposition 5.1.1, $\Gamma \vdash \{A\}P\{B\}$ is true. □

We will give an unsound example for the naive logical system obtained by taking the union of axioms and inference rules from Hoare's logic for recursive procedures and separation logic.

Definition 5.1.9 *The axiom (*Axiom 9: Invariance Axiom*) has been defined in [3] as:*

$$\frac{}{\vdash \{A\}P\{A\}}\,(invariance) \quad (FV(A) \cap EFV(P) = \emptyset)$$

Definition 5.1.10 *We define the logical system 'Separation+Invariance Logic' as the logical system obtained from the separation logic by adding the axiom (invariance).*

Proposition 5.1.11 *The axiom (invariance) is not sound in Separation+Invariance Logic.*

Proof. $\vdash \{\mathrm{emp}\}x := \mathrm{cons}(e_1, e_2)\{\mathrm{emp}\}$ is provable by the axiom (invariance) in the system Separation+Invariance Logic since $\mathrm{FV}(\mathrm{emp}) \cap \mathrm{EFV}(x := \mathrm{cons}(e_1, e_2)) = \emptyset$. However it is apparently false. □

5.2 Expressiveness

5.2.1 Coding of Assertions

This section proves the expressiveness theorem. Our technique is to extend the expressiveness theorem given in [19] to mutual recursive procedure calls. In this section, we first assume that for given assertions and programs, we fix some sequence $\overrightarrow{x}$ of variables that contains the free variables of the assertions and the extended free variables of the programs. We will next define the formulas $\mathrm{Store}_{\overrightarrow{x}}(m)$ and $\mathrm{Heap}(m)$ for describing the current store and the current heap respectively. Next we will provide the *pure* formulas $\mathrm{EEval}_{e,\overrightarrow{x}}(n,k)$ and $\mathrm{BEval}_{A,\overrightarrow{x}}(n)$, which express the meaning of the expression e and the *pure* formula A respectively. Then we will define the *pure* formula $\mathrm{HEval}_A(m)$ for expressing the meaning of the assertion A at the heap by m. By using it, we will define the *pure* formula $\mathrm{Eval}_{A,\overrightarrow{x}}(n,m)$, which expresses the meaning of the assertion A. We will also define the *pure* formula $\mathrm{Exec}_{P,\overrightarrow{x}}(n,m)$ for the meaning of the program P. Finally we will define the formula $W_{P,A}(\overrightarrow{x})$ for the weakest precondition of the program P and the assertion A, and we will prove the expressiveness theorem that states $W_{P,A}(\overrightarrow{x})$ indeed expresses the weakest precondition.

We assume a standard surjective pairing function on natural numbers. For natural numbers n, m, we will write (n,m) to denote the number that represents the pair of n and m. We also assume a standard surjective coding of a sequence of natural numbers by a natural number. We will write $\langle n_1,\ldots,n_k\rangle$ for the number that represents the sequence $n_1,\ldots,n_k$. When the number n represents a sequence, $lh(n)$ and $(n)_i$ denote the length of the sequence and the i-th element of the sequence respectively.

The following predicates for handling sequences are known to be definable in the language of Peano arithmetic. $\mathrm{Pair}(k,n,m)$ is defined to hold if k is the number that represents the pair of n and m. $\mathrm{Lh}(n,k)$ is defined to hold if k is the length of the sequence represented by n. That is, $\mathrm{Lh}(\langle n_1,\ldots,n_k\rangle,k)$ holds. $\mathrm{Elem}(n,i,k)$ is defined to hold if k is the i-th element in the sequence represented by n. That is, $\mathrm{Elem}(\langle n_1,\ldots,n_k\rangle,i-1,n_i)$ holds.

We code the piece of the store s for variables $x_1, \ldots, x_k$ by the number $\langle n_1, \ldots, n_k \rangle$ where $s(x_i) = n_i$. We code the heap h by the number $\langle m_1, \ldots, m_k \rangle$ where $\mathrm{Dom}(h) = \{n_1, \ldots, n_k\}$, $0 < n_1 < \ldots < n_k$ and m_i is the number that represents the pair of n_i and $h(n_i)$. We code the result of a program execution by coding abort and (s, h) by 0 and $k + 1$ respectively where the piece of s is coded by n, h is coded by m, and k is the pair of numbers n and m. The number that represents a given heap is unique. For example, the number $\langle (1, 5), (3, 8) \rangle$ represents the heap h such that $\mathrm{Dom}(h) = \{1, 3\}$ and $h(1) = 5$, $h(3) = 8$. Note that for heap representation we do not think the numbers $\langle (3, 8), (1, 5) \rangle$ or $\langle (1, 5), (1, 5), (3, 8) \rangle$ since $n_1 < n_2$ is violated.

We say A is true at (s, h) when $[\![A]\!]_{(s,h)} = \mathrm{True}$. The formula $A \leftrightarrow B$ is defined as $(A \to B) \wedge (B \to A)$.

$\emptyset$ sometimes denotes the empty heap, that is, $\emptyset(x)$ is undefined for all $x \in N$.

We define the following *pure* formulas. First we define coding of our assertion language. It is the same as [19].

$$
\begin{aligned}
\mathrm{Lesslh}(i, n) &= \exists x(\mathrm{Lh}(n, x) \wedge i < x), \\
\mathrm{Addseq}(k, n, m) &= \exists x(\mathrm{Lh}(n, x) \wedge \mathrm{Lh}(m, x + 1)) \wedge \mathrm{Elem}(m, 0, k) \wedge \\
&\quad \forall yx(\mathrm{Lesslh}(y, n) \wedge \mathrm{Elem}(n, y, x) \to \mathrm{Elem}(m, y + 1, x)).
\end{aligned}
$$

The predicate $\mathrm{Lesslh}(i, n)$ means $i < lh(n)$. The predicate $\mathrm{Addseq}(k, n, m)$ means $\langle k \rangle \cdot n = m$ where $\cdot$ denotes the concatenation of sequences.

Definition 5.2.1 *We define the formulas $Store_{x_1,\ldots,x_n}(m)$ and $Heap(m)$.*

$$
\begin{aligned}
Store_{x_1,\ldots,x_n}(m) &= Lh(m, n) \wedge Elem(m, 0, x_1) \wedge \ldots \wedge Elem(m, n - 1, x_n), \\
Lookup(m, l, k) &= \exists yz(Lesslh(y, m) \wedge Elem(m, y, z) \wedge Pair(z, l, k)), \\
IsHeap(m) &= \forall i x_1 y_1 z_1 x_2 y_2 z_2 (Lesslh(i + 1, m) \wedge Elem(m, i, x_1) \wedge \\
&\quad Pair(x_1, y_1, z_1) \wedge Elem(m, i + 1, x_2) \wedge Pair(x_2, y_2, z_2) \to \\
&\quad 0 < y_1 \wedge y_1 < y_2), \\
Heap(m) &= IsHeap(m) \wedge \forall xy(Lookup(m, x, y) \leftrightarrow (x \mapsto y * True)).
\end{aligned}
$$

The predicate $\mathrm{Store}_{x_1,\ldots,x_n}(\langle m_1, \ldots, m_n \rangle)$ means $s(x_i) = m_i$ where s is the current store. The predicate $\mathrm{Lookup}(m, l, k)$ means $h(l) = k$ where m represents the heap h. The predicate IsHeap is defined so that $\mathrm{IsHeap}(m)$ means that there is some heap that the number m represents. The predicate $\mathrm{Heap}(\langle(l_1, n_1), \ldots, (l_k, n_k)\rangle)$ means $\mathrm{Dom}(h) = \{l_1, \ldots, l_k\}$, $0 < l_1 < \ldots < l_k$ and $h(l_i) = n_i$ where h is the current heap.

Definition 5.2.2 *We define the pure formulas $EEval_{e,\overrightarrow{x}}(n, k)$ for the expression e and $BEval_{A,\overrightarrow{x}}(n)$ for the pure formula A where we suppose $\overrightarrow{x}$ includes $FV(e)$ and $FV(A)$ respectively.*

$$\begin{array}{rcl} EEval_{e,\overrightarrow{x}}(n, k) & = & \exists\overrightarrow{x}(Store_{\overrightarrow{x}}(n) \wedge e = k), \\ BEval_{A,\overrightarrow{x}}(n) & = & \exists\overrightarrow{x}(Store_{\overrightarrow{x}}(n) \wedge A). \end{array}$$

$EEval_{e,\overrightarrow{x}}(n, k)$ means $[\![e]\!]_s = k$ where n represents the store s. $BEval_{A,\overrightarrow{x}}(n)$ means $[\![A]\!]_s =$ True where n represents the store s.

We define the following *pure* formulas.

$$\begin{array}{rcl} \mathrm{Pair2}(z, x, y) & = & \exists w(z = w + 1 \wedge \mathrm{Pair}(w, x, y) \wedge \mathrm{IsHeap}(y)), \\ \mathrm{Domain}(k, m) & = & \exists y \mathrm{Lookup}(m, k, y), \\ \mathrm{Separate}(m, m_1, m_2) & = & \mathrm{IsHeap}(m) \wedge \mathrm{IsHeap}(m_1) \wedge \mathrm{IsHeap}(m_2) \wedge \\ & & \forall x(\exists y(\mathrm{Elem}(m, y, x)) \leftrightarrow \exists y(\mathrm{Elem}(m_1, y, x) \vee \\ & & \mathrm{Elem}(m_2, y, x))) \wedge \forall x_1 x_2 y_1 y_2(\mathrm{Lookup}(m_1, x_1, y_1) \wedge \\ & & \mathrm{Lookup}(m_2, x_2, y_2) \rightarrow x_1 \neq x_2). \end{array}$$

$\mathrm{Domain}(k, m)$ means $k \in Dom(h)$ where m represents the heap h. $\mathrm{Separate}(m, m_1, m_2)$ means $h = h_1 + h_2$ where m, m_1, and m_2 represent the heaps h, h_1, and h_2 respectively.

Definition 5.2.3 *We define the pure formula $HEval_A(x)$ for the assertion A by induction* *on A.*

$$\begin{aligned}
HEval_A(m) &= A \qquad (A \text{ is a pure formula}),\\
HEval_{emp}(m) &= \neg\exists xy Lookup(m,x,y),\\
HEval_{e_1\mapsto e_2}(m) &= e_1 > 0 \wedge \forall xy(Lookup(m,x,y) \leftrightarrow x = e_1 \wedge y = e_2),\\
HEval_{\neg A}(m) &= \neg HEval_A(m),\\
HEval_{A\wedge B}(m) &= HEval_A(m) \wedge HEval_B(m),\\
HEval_{A\vee B}(m) &= HEval_A(m) \vee HEval_B(m),\\
HEval_{A\to B}(m) &= HEval_A(m) \to HEval_B(m),\\
HEval_{\forall xA}(m) &= \forall x HEval_A(m),\\
HEval_{\exists xA}(m) &= \exists x HEval_A(m),\\
HEval_{A*B}(m) &= \exists y_1 y_2(Separate(m,y_1,y_2) \wedge HEval_A(y_1) \wedge HEval_B(y_2)),\\
HEval_{A-*B}(m) &= \forall y_1 y_2(HEval_A(y_2) \wedge Separate(y_1,m,y_2) \to HEval_B(y_1)).
\end{aligned}$$

$\mathrm{HEval}_A(m)$ means $[\![A]\!]_{(s,h)} = \text{True}$ where s is the current store and m represents the heap h.

Definition 5.2.4 *We define the pure formula $Eval_{A,\overrightarrow{x}}(n,m)$ for the assertion A. We suppose $\overrightarrow{x}$ includes $FV(A)$.*

$$Eval_{A,\overrightarrow{x}}(n,m) = \exists\overrightarrow{x}(Store_{\overrightarrow{x}}(n) \wedge IsHeap(m) \wedge HEval_A(m)).$$

$\mathrm{Eval}_{A,\overrightarrow{x}}(n,m)$ means $[\![A]\!]_{(s,h)} = \text{True}$ where n represents the store s and m represents the heap h.

5.2.2 Coding of Programs

We define the following *pure* formulas.

$$\begin{aligned}
&\mathrm{New2}(n,m) = n > 0 \wedge \neg\mathrm{Domain}(n,m) \wedge \neg\mathrm{Domain}(n+1,m),\\
&\mathrm{ChangeStore}_{x_0,\ldots,x_n,x_i}(m_1,k,m_2) = \mathrm{Lh}(m_1,n+1) \wedge \mathrm{Lh}(m_2,n+1)\wedge\\
&\qquad \forall yx(y < n+1 \wedge y \neq i \wedge \mathrm{Elem}(m_1,y,x) \to \mathrm{Elem}(m_2,y,x))\wedge\\
&\qquad \mathrm{Elem}(m_2,i,k),\\
&\mathrm{ChangeHeap}(m_1,l,k,m_2) = \forall xy(x \neq l \to (\mathrm{Lookup}(m_1,x,y) \leftrightarrow\\
&\qquad \mathrm{Lookup}(m_2,x,y))) \wedge \mathrm{Lookup}(m_2,l,k).
\end{aligned}$$

$\mathrm{New2}(n,m)$ means n is the address of free cells in h where m represents the heap h. That is, the address n can be used by the next $x := \mathrm{cons}(e_1,e_2)$ statement. $\mathrm{ChangeStore}_{x_0,\ldots,x_n,x_i}(m_1,k,m_2)$ means m_2 represents the store $s[x_i := k]$ where m_1 represents the store s. $\mathrm{ChangeHeap}(m_1,l,k,m_2)$ means m_2 represents the heap $h[l := k]$ where m_1 represents the heap h.

We say the number n represents the result r if $r = \mathrm{abort}$ and $n = 0$ or $r = (s,h)$ and $n = (m,k)+1$ where m represents the store s and k represents the heap h.

Next we extend coding of programs used in [19] to mutual recursive procedure calls.

Definition 5.2.5 *We define the pure formula $ExecU_{P,\vec{x}}(m,n_1,n_2)$ by induction on (m,P) in Figure 5.2.1. We define*

$$Exec_{P,\vec{x}}(n_1,n_2) \quad = \quad \exists k(ExecU_{P,\vec{x}}(k,n_1,n_2))$$

$\mathrm{ExecU}_{P,\vec{x}}(k,n_1,n_2)$ is true if and only if the following: when we execute the k-level unfolding $P^{(k)}$ of the program P from the state coded by n_1, one of the possible resulting states is the state coded by n_2. The predicate $\mathrm{Exec}_{P,\vec{x}}(n_1,n_2)$ means $[\![P]\!](r_1) \ni r_2$ where n_1 and n_2 represent r_1 and r_2 respectively.

$$
\begin{aligned}
&\mathrm{ExecU}_{x:=e,\vec{x}}(m,n_1,n_2) = (n_1 = 0 \to n_2 = 0)\wedge\\
&\qquad (n_1 > 0 \to \exists y_1 z_1 y_2 w(\mathrm{Pair2}(n_1,y_1,z_1) \wedge \mathrm{EEval}_{e,\vec{x}}(y_1,w)\wedge\\
&\qquad \mathrm{ChangeStore}_{\vec{x},x}(y_1,w,y_2) \wedge \mathrm{Pair2}(n_2,y_2,z_1))),\\
&\mathrm{ExecU}_{\mathrm{if}\,(b)\,\mathrm{then}\,(P_1)\,\mathrm{else}\,(P_2),\vec{x}}(m,n_1,n_2) = (n_1 = 0 \to n_2 = 0)\wedge\\
&\qquad (n_1 > 0 \to \exists xy(\mathrm{Pair2}(n_1,x,y) \wedge (\mathrm{BEval}_{b,\vec{x}}(x)\to\\
&\qquad \mathrm{ExecU}_{P_1,\vec{x}}(m,n_1,n_2)) \wedge (\neg\mathrm{BEval}_{b,\vec{x}}(x) \to \mathrm{ExecU}_{P_2,\vec{x}}(m,n_1,n_2)))),\\
&\mathrm{ExecU}_{\mathrm{while}\,(b)\,\mathrm{do}\,(P),\vec{x}}(m,n_1,n_2) = (n_1 = 0 \to n_2 = 0)\wedge\\
&\qquad (n_1 > 0 \to \exists wz(\mathrm{Lh}(w,z+1) \wedge \mathrm{Elem}(w,0,n_1) \wedge \mathrm{Elem}(w,z,n_2)\wedge\\
&\qquad \forall w_1(w_1 < z \to \exists z_1 z_2 w_2 w_3(\mathrm{Elem}(w,w_1,z_1) \wedge \mathrm{Elem}(w,w_1+1,z_2)\wedge\\
&\qquad z_1 > 0 \wedge \mathrm{Pair2}(z_1,w_2,w_3) \wedge \mathrm{BEval}_{b,\vec{x}}(w_2) \wedge \mathrm{ExecU}_{P,\vec{x}}(m,z_1,z_2))))\\
&\qquad \wedge(n_2 > 0 \to \exists yz(\mathrm{Pair2}(n_2,y,z) \wedge \neg\mathrm{BEval}_{b,\vec{x}}(y)))),\\
&\mathrm{ExecU}_{P_1;P_2,\vec{x}}(m,n_1,n_2) = \exists z(\mathrm{ExecU}_{P_1,\vec{x}}(m,n_1,z) \wedge \mathrm{ExecU}_{P_2,\vec{x}}(m,z,n_2)),\\
&\mathrm{ExecU}_{\mathrm{skip},\vec{x}}(m,n_1,n_2) = (n_1 = n_2),\\
&\mathrm{ExecU}_{x:=\mathrm{cons}(e_1,e_2),\vec{x}}(m,n_1,n_2) = (n_1 = 0 \to n_2 = 0)\wedge\\
&\qquad (n_1 > 0 \to \exists y_1 z_1 y_2 z_2 w w_1 w_2(\mathrm{Pair2}(n_1,y_1,z_1) \wedge \mathrm{EEval}_{e_1,\vec{x}}(y_1,w_1)\wedge\\
&\qquad \mathrm{EEval}_{e_2,\vec{x}}(y_1,w_2) \wedge \mathrm{New2}(w,z_1) \wedge \mathrm{ChangeStore}_{\vec{x},x}(y_1,w,y_2)\wedge\\
&\qquad \forall xy(x \neq w \wedge x \neq w+1 \to (\mathrm{Lookup}(z_1,x,y) \leftrightarrow \mathrm{Lookup}(z_2,x,y)))\wedge\\
&\qquad \mathrm{Lookup}(z_2,w,w_1) \wedge \mathrm{Lookup}(z_2,w+1,w_2) \wedge \mathrm{Pair2}(n_2,y_2,z_2))),\\
&\mathrm{ExecU}_{x:=[e],\vec{x}}(m,n_1,n_2) = (n_1 = 0 \to n_2 = 0)\wedge\\
&\qquad (n_1 > 0 \to \exists y_1 z_1 y_2 w w_1(\mathrm{Pair2}(n_1,y_1,z_1) \wedge \mathrm{EEval}_{e,\vec{x}}(y_1,w)\wedge\\
&\qquad (\neg\mathrm{Domain}(w,z_1) \to n_2 = 0) \wedge (\mathrm{Domain}(w,z_1)\to\\
&\qquad \mathrm{Lookup}(z_1,w,w_1) \wedge \mathrm{ChangeStore}_{\vec{x},x}(y_1,w_1,y_2) \wedge \mathrm{Pair2}(n_2,y_2,z_1)))),\\
&\mathrm{ExecU}_{[e_1]:=e_2,\vec{x}}(m,n_1,n_2) = (n_1 = 0 \to n_2 = 0)\wedge\\
&\qquad (n_1 > 0 \to \exists y_1 z_1 z_2 w_1 w_2(\mathrm{Pair2}(n_1,y_1,z_1) \wedge \mathrm{EEval}_{e_1,\vec{x}}(y_1,w_1)\wedge\\
&\qquad \mathrm{EEval}_{e_2,\vec{x}}(y_1,w_2) \wedge (\neg\mathrm{Domain}(w_1,z_1) \to n_2 = 0)\wedge\\
&\qquad (\mathrm{Domain}(w_1,z_1) \to \mathrm{ChangeHeap}(z_1,w_1,w_2,z_2) \wedge \mathrm{Pair2}(n_2,y_1,z_2)))),\\
&\mathrm{ExecU}_{\mathrm{dispose}(e),\vec{x}}(m,n_1,n_2) = (n_1 = 0 \to n_2 = 0)\wedge\\
&\qquad (n_1 > 0 \to \exists y_1 z_1 z_2 w(\mathrm{Pair2}(n_1,y_1,z_1) \wedge \mathrm{EEval}_{e,\vec{x}}(y_1,w)\wedge\\
&\qquad (\neg\mathrm{Domain}(w,z_1) \to n_2 = 0) \wedge (\mathrm{Domain}(w,z_1)\to\\
&\qquad \forall xy(\mathrm{Lookup}(z_1,x,y) \wedge x \neq w \leftrightarrow \mathrm{Lookup}(z_2,x,y)) \wedge \mathrm{Pair2}(n_2,y_1,z_2)))),\\
&\mathrm{ExecU}_{R_i,\vec{x}}(0,n_1,n_2) = (n_1 = 0 \wedge n_2 = 0),\\
&\mathrm{ExecU}_{R_i,\vec{x}}(k+1,n_1,n_2) = \mathrm{ExecU}_{Q_i,\vec{x}}(k,n_1,n_2).
\end{aligned}
$$

Figure 5.2.1: Definition of ExecU

We define the following abbreviations. Note that they are not formulas.

$$\begin{aligned}
\text{Storecode}_{x_1,\dots,x_n}(m,s) &= \text{Lh}(m,n) \wedge \forall i < n(\text{Elem}(m,i,s(x_{i+1}))),\\
\text{Heapcode}(m,h) &= \text{IsHeap}(m) \wedge \forall ln(h(l) = n \leftrightarrow \text{Lookup}(m,l,n)),\\
\text{Result}_{\vec{x}}(n,r) &= n = 0 \wedge r = \text{abort} \vee\\
&\ n > 0 \wedge \exists shyz(r = (s,h) \wedge \text{Pair2}(n,y,z) \wedge\\
&\ \text{Storecode}_{\vec{x}}(y,s) \wedge \text{Heapcode}(z,h)).
\end{aligned}$$

$\text{Storecode}_{x_1,\dots,x_n}(m,s)$ means that the number m is the code that represents the store s for variables $x_1,\dots,x_n$. $\text{Heapcode}(m,h)$ means the number m is the code that represents the heap h. $\text{Result}_{\vec{x}}(n,r)$ means the number n represents the result r.

The following lemma says that $\text{ExecU}_{P,\vec{x}}(k,n_1,n_2)$ simulates the execution of the k level of unfolding of the program P.

Lemma 5.2.6 *(1) $ExecU_{\Omega,\vec{x}}(0,n_1,n_2)$ is true if and only if $n_1 = n_2 = 0$.*

(2) $ExecU_{P,\vec{x}}(k,n_1,n_2)$ is true if and only if $ExecU_{P^{(k)},\vec{x}}(0,n_1,n_2)$ is true.

(3) $ExecU_{P,\vec{x}}(k,n_1,n_2)$ is true if and only if $Exec_{P^{(k)},\vec{x}}(n_1,n_2)$ is true.

Proof. (1) We will show from the left-hand side to the right-hand side. Assume that $ExecU_{\Omega,\vec{x}}(0,n_1,n_2)$ is true. By definition, $ExecU_{\text{while }(0=0)\text{ do (skip)},\vec{x}}(0,n_1,n_2)$ is true.

Case 1. $n_1 = 0$. Then by definition, $n_2 = 0$.

Case 2. $n_1 > 0$. Then $\exists wz(\text{Lh}(w,z+1) \wedge \text{Elem}(w,0,n_1) \wedge \text{Elem}(w,z,n_2) \wedge \forall w_1(w_1 < z \to \exists z_1 z_2 w_2 w_3(\text{Elem}(w,w_1,z_1) \wedge \text{Elem}(w,w_1+1,z_2) \wedge z_1 > 0 \wedge \text{Pair2}(z_1,w_2,w_3) \wedge \text{BEval}_{0=0,\vec{x}}(w_2) \wedge \text{ExecU}_{\text{skip},\vec{x}}(0,z_1,z_2)))) \wedge (n_2 > 0 \to \exists yz(\text{Pair2}(n_2,y,z) \wedge \neg\text{BEval}_{b,\vec{x}}(y)))$ holds.

Case 2.1. $n_2 > 0$. Since $\text{BEval}_{0=0,\vec{x}}(y)$ is true, $\exists yz(\text{Pair2}(n_2,y,z) \wedge \neg\text{BEval}_{0=0,\vec{x}}(y))$ is false. Hence we do not have this case.

Case 2.2. $n_2 = 0$. We have $w = \langle w_1,\dots,w_n\rangle$ and $w_1 = n_1, w_n = n_2$ and for all $1 \le i < n$, $\text{ExecU}_{\text{skip},\vec{x}}(0,w_i,w_{i+1})$ is true. Then for all $1 \le i < n$, $w_i = w_{i+1}$. Since, $n_2 = 0$, we have $n_1 = 0$. It contradicts the assumption.

Therefore, $n_1 = n_2 = 0$.

The opposite direction can be shown directly by definition.

(2) Proved by induction on (k, P). We will consider the cases of k.

Case 1. $k = 0$.

Here only important case is when P is R_i. Because, induction hypothesis proves other cases in a similar way to those in Case 2.

Case 1.1. P is R_i.

By definition, $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is $n_1 = n_2 = 0$. By Proposition 4.1.10 (2), $\mathrm{ExecU}_{P^{(k)},\overrightarrow{x}}(0, n_1, n_2)$ is $\mathrm{ExecU}_{\Omega,\overrightarrow{x}}(0, n_1, n_2)$. By (1), they are equivalent.

Case 2. $k = k_1 + 1$.

Case 2.1. P is *atomic*.

P is $P^{(k)}$, by definition.

Since $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ does not depend on k, $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is the same as $\mathrm{ExecU}_{P^{(k)},\overrightarrow{x}}(0, n_1, n_2)$.

Case 2.2. P is if (b) then (P_1) else (P_2).

Suppose we have some x and y such that $\mathrm{Pair2}(n_1, x, y)$ is true. $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is equivalent to $\mathrm{ExecU}_{P_1,\overrightarrow{x}}(k, n_1, n_2)$ when $\mathrm{BEval}_{b,\overrightarrow{x}}(x)$ is true and $\mathrm{ExecU}_{P_2,\overrightarrow{x}}(k, n_1, n_2)$ when $\neg\mathrm{BEval}_{b,\overrightarrow{x}}(x)$ is true. By induction hypothesis, it is equivalent to $\mathrm{ExecU}_{P_1^{(k)},\overrightarrow{x}}(0, n_1, n_2)$ when $\mathrm{BEval}_{b,\overrightarrow{x}}(x)$ is true and $\mathrm{ExecU}_{P_2^{(k)},\overrightarrow{x}}(0, n_1, n_2)$ when $\neg\mathrm{BEval}_{b,\overrightarrow{x}}(x)$ is true. Hence it is equivalent to $\mathrm{ExecU}_{P^{(k)},\overrightarrow{x}}(0, n_1, n_2)$.

Case 2.3. P is $P_1; P_2$.

$\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is true if and only if $\mathrm{ExecU}_{P_1,\overrightarrow{x}}(k, n_1, n_3)$ is true and $\mathrm{ExecU}_{P_1,\overrightarrow{x}}(k, n_3, n_2)$ is true for some n_3. By induction hypothesis, it equivalent to the fact that $\mathrm{ExecU}_{P_1^{(k)},\overrightarrow{x}}(0, n_1, n_3)$ is true and $\mathrm{ExecU}_{P_2^{(k)},\overrightarrow{x}}(0, n_3, n_2)$ is true. Therefore, it is equivalent to $\mathrm{ExecU}_{P^{(k)},\overrightarrow{x}}(0, n_1, n_2)$.

Case 2.4. P is while (b) do (P_1).

$\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is true if and only if $\mathrm{ExecU}_{P_1,\overrightarrow{x}}(k, m_i, m_{i+1})$ is true for all $1 \leq i < l$ for some l, where $m_1 = n_1$ and $m_l = n_2$ by definition. By induction hypothesis, it is equivalent to $\mathrm{ExecU}_{P_1^{(k)},\overrightarrow{x}}(0, m_i, m_{i+1})$ for all such i. Therefore, $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is true if and only if $\mathrm{ExecU}_{P^{(k)},\overrightarrow{x}}(0, n_1, n_2)$ is true.

Case 2.5. P is R_i.

By definition $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is $\mathrm{ExecU}_{Q_i,\overrightarrow{x}}(k_1, n_1, n_2)$. By induction hypothesis, it is equivalent to $\mathrm{ExecU}_{Q_i^{(k_1)},\overrightarrow{x}}(0, n_1, n_2)$. Since $R_i^{(k)} = R_i[\overrightarrow{Q^{(k_1)}}] = Q_i^{(k_1)}$ by definition, $\mathrm{ExecU}_{R_i^{(k)},\overrightarrow{x}}(0, n_1, n_2)$ is $\mathrm{ExecU}_{Q_i^{(k_1)},\overrightarrow{x}}(0, n_1, n_2)$.

Therefore, $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is equivalent to $\mathrm{ExecU}_{R_i^{(k)},\overrightarrow{x}}(0, n_1, n_2)$.

(3) From the left-hand side to the right-hand side. Assume the left-hand side. By (2), $\mathrm{ExecU}_{P^{(k)},\overrightarrow{x}}(0, n_1, n_2)$ is true. Hence $\mathrm{Exec}_{P^{(k)},\overrightarrow{x}}(n_1, n_2)$ is true.

From the right-hand side to the left-hand side. Assume the right-hand side. Then $\mathrm{ExecU}_{P^{(k)},\overrightarrow{x}}(m, n_1, n_2)$ is true for some m. It is the same as $\mathrm{ExecU}_{P^{(k)},\overrightarrow{x}}(0, n_1, n_2)$. Therefore, by (2), $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is true. $\square$

5.2.3 Representation Lemma for Assertions

The next lemma shows that the *pure* formulas $\mathrm{EEval}_{e,\overrightarrow{x}}(n, k)$, $\mathrm{BEval}_{A,\overrightarrow{x}}(n)$, $\mathrm{HEval}_A(m)$ and $\mathrm{Eval}_{A,\overrightarrow{x}}(n, m)$ actually have the meaning we explained above. The next lemma can be proved in the same way as [19].

Lemma 5.2.7 (Representation of Assertions) *(1) $EEval_{e,\overrightarrow{x}}(n, k)$ is true if and only if $\exists s(Storecode_{\overrightarrow{x}}(n, s) \wedge [\![e]\!]_s = k)$ holds.*

(2) $BEval_{A,\overrightarrow{x}}(n)$ is true if and only if $\exists s(Storecode_{\overrightarrow{x}}(n, s) \wedge [\![A]\!]_s = True)$ holds.

(3) If $Heapcode(m, h)$ holds then $[\![HEval_A(m)]\!]_s = [\![A]\!]_{(s,h)}$ also holds.

(4) $Eval_{A,\overrightarrow{x}}(n, m)$ is true if and only if $\exists sh(Storecode_{\overrightarrow{x}}(n, s) \wedge Heapcode(m, h) \wedge [\![A]\!]_{(s,h)} = True)$ holds.

Proof. (1) This is similarly proved to (2).

(2) The left-hand side is equivalent to $\forall s([\![\exists\vec{x}(\mathrm{Store}_{\vec{x}}(n) \wedge A)]\!]_s = \mathrm{True})$. It is equivalent to $\exists s([\![\mathrm{Store}_{\vec{x}}(n) \wedge A]\!]_s = \mathrm{True})$. Hence it is equivalent to $\exists s([\![\mathrm{Store}_{\vec{x}}(n)]\!]_s = \mathrm{True} \wedge [\![A]\!]_s = \mathrm{True})$. Since $[\![\mathrm{Store}_{\vec{x}}(n)]\!]_s = \mathrm{True}$ is equivalent to $\mathrm{Storecode}_{\vec{x}}(n, s)$, we have the claim.

(3) By induction on A, we will show that $\forall mh(\mathrm{Heapcode}(m, h) \to ([\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True} \leftrightarrow [\![A]\!]_{(s,h)} = \mathrm{True}))$ holds. Assume that $\mathrm{Heapcode}(m, h)$ holds. We will show that $[\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True} \leftrightarrow [\![A]\!]_{(s,h)} = \mathrm{True}$ holds. We consider cases according to A.

Case 1. A is a *pure* formula. We have $\mathrm{HEval}_A(m) = A$ and the claim holds.

Case 2. $A = \mathrm{emp}$. By definition $\mathrm{HEval}_A(m) = \neg\exists xy\mathrm{Lookup}(m, x, y)$. By definition, $[\![\mathrm{emp}]\!]_{(s,h)} = \mathrm{True}$ if and only if $\mathrm{Dom}(h) = \emptyset$. $\mathrm{Dom}(h) = \emptyset$ if and only if $\neg\exists xy\mathrm{Lookup}(m, x, y)$ is true. Hence we have the claim.

Case 3. $A = e_1 \mapsto e_2$. Let k_i be $[\![e_i]\!]_s$. All of $[\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True}$, $k_1 > 0 \wedge \forall xy(\mathrm{Lookup}(m, x, y) \leftrightarrow x = k_1 \wedge y = k_2)$, $h = \emptyset[k_1 := k_2]$, and $[\![A]\!]_{(s,h)} = \mathrm{True}$ are equivalent. Hence the claim holds.

Case 4. $A = A_1 * A_2$.

From the left-hand side to the right-hand side. Assume $[\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True}$. We will show $[\![A]\!]_{(s,h)} = \mathrm{True}$. We have $[\![\mathrm{Separate}(m, y_1, y_2) \wedge \mathrm{HEval}_{A_1}(y_1) \wedge \mathrm{HEval}_{A_2}(y_2)]\!]_{s[y_1:=m_1, y_2:=m_2]} = \mathrm{True}$ for some m_1, m_2. Then $[\![\mathrm{HEval}_{A_i}(m_i)]\!]_s = \mathrm{True}$. We have h_i such that $\mathrm{Heapcode}(m_i, h_i)$ holds. Then $h = h_1 + h_2$. By induction hypothesis with $[\![\mathrm{HEval}_{A_i}(m_i)]\!]_s = \mathrm{True}$, we have $[\![A_i]\!]_{(s,h_i)} = \mathrm{True}$. Hence $[\![A]\!]_{(s,h)} = \mathrm{True}$.

From the right-hand side to the left-hand side. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. We will show $[\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True}$. There are h_1, h_2 such that $h = h_1 + h_2$ and $[\![A_i]\!]_{(s,h_i)} = \mathrm{True}$. We have m_1, m_2 such that $\mathrm{Heapcode}(m_i, h_i)$ holds. Then $\mathrm{Separate}(m, m_1, m_2)$ is true. By induction hypothesis for A_i, we have $[\![\mathrm{HEval}_{A_i}(m_i)]\!]_s = \mathrm{True}$. Hence $[\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True}$ holds by taking $y_1 = m_1$ and $y_2 = m_2$.

Case 5. $A = A_1 \mathrel{-\!\!*} A_2$.

From the left-hand side to the right-hand side. Assume $[\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True}$. We will show $[\![A]\!]_{(s,h)} = \mathrm{True}$. Assume $[\![A_1]\!]_{(s,h_1)} = \mathrm{True}$ and $h + h_1$ exists. We will show $[\![A_2]\!]_{(s,h+h_1)} = \mathrm{True}$. We have m_1, m_2 such that $\mathrm{Heapcode}(m_2, h_1)$ and $\mathrm{Heapcode}(m_1, h + h_1)$ hold. By induction hypothesis for A_1, we have $[\![\mathrm{HEval}_{A_1}(m_2)]\!]_s = \mathrm{True}$. We also have $\mathrm{Separate}(m_1, m, m_2)$ is true. From the assumption, we have $[\![\mathrm{HEval}_{A_2}(m_1)]\!]_s = \mathrm{True}$. By induction hypothesis for A_2, we have $[\![A_2]\!]_{(s,h+h_1)} = \mathrm{True}$.

From the right-hand side to the left-hand side. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. We will show $[\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True}$. Fix m_1, m_2 and assume $[\![\mathrm{HEval}_{A_1}(m_2) \wedge \mathrm{Separate}(m_1, m, m_2)]\!]_s = \mathrm{True}$. We will show $[\![\mathrm{HEval}_{A_2}(m_1)]\!]_s = \mathrm{True}$. We have h_1, h_2 such that $\mathrm{Heapcode}(m_1, h_1)$ and $\mathrm{Heapcode}(m_2, h_2)$ hold. Then $h_1 = h + h_2$. By induction hypothesis for A_1, we have $[\![A_1]\!]_{(s,h_2)} = \mathrm{True}$. From the assumption, we have $[\![A_2]\!]_{(s,h_1)} = \mathrm{True}$. By induction hypothesis for A_2, we have $[\![\mathrm{HEval}_{A_2}(m_1)]\!]_s = \mathrm{True}$.

Cases $A = \neg A_1, A_1 \wedge A_2, A_1 \vee A_2, A_1 \to A_2, \forall x A_1, \exists x A_1$ are proved straightforwardly by using induction hypothesis.

(4) The right-hand side is equivalent to $\exists h(\mathrm{Heapcode}(m, h) \wedge \exists s(\mathrm{Storecode}_{\vec{x}}(n, s) \wedge [\![A]\!]_{(s,h)} = \mathrm{True}))$. Since $[\![A]\!]_{(s,h)} = [\![\mathrm{HEval}_A(m)]\!]_s$ under $\mathrm{Heapcode}(m, h)$ by (3), it is equivalent to $\exists h(\mathrm{Heapcode}(m, h) \wedge \exists s(\mathrm{Storecode}_{\vec{x}}(n, s) \wedge [\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True}))$. It is equivalent to $\exists s(\mathrm{IsHeap}(m) \wedge \mathrm{Storecode}_{\vec{x}}(n, s) \wedge [\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True})$. It can be shown from (2) that $\mathrm{BEval}_{\mathrm{HEval}_A(m), \vec{x}}(n) = \mathrm{True}$ if and only if $\exists s(\mathrm{Storecode}_{\vec{x}}(n, s) \wedge [\![\mathrm{HEval}_A(m)]\!]_s = \mathrm{True})$ holds. Hence it is equivalent to $\mathrm{IsHeap}(m) \wedge \mathrm{BEval}_{\mathrm{HEval}_A(m), \vec{x}}(n)$. By definition, it is equivalent to $\exists \vec{x}(\mathrm{IsHeap}(m) \wedge \mathrm{Store}_{\vec{x}}(n) \wedge \mathrm{HEval}_A(m))$, which is the left-hand side by the definition of $\mathrm{Eval}_{A, \vec{x}}$. $\square$

5.2.4 Representation Lemma for Programs

The next lemma shows that the *pure* formula $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ actually have the meaning we explained above.

Lemma 5.2.8 (Representation of Programs) *(1) If $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is true, then for all r_1 such that $\mathrm{Result}_{\overrightarrow{x}}(n_1, r_1)$, we have r_2 such that $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$ and $[\![P^{(k)}]\!]^-(r_1) \ni r_2$.*

(2) If $[\![P^{(k)}]\!]^-(r_1) \ni r_2$, $\mathrm{Result}_{\overrightarrow{x}}(n_1, r_1)$, and $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$ hold, then $\mathrm{ExecU}_{P,\overrightarrow{x}}(k, n_1, n_2)$ is true.

Proof. (1) We will prove it by induction on (k, P). We will consider the cases of P.

Case 1. P is $x := e$.

Assume that $\mathrm{ExecU}_{x:=e,\overrightarrow{x}}(k, n_1, n_2)$ is true and r_1 is given. If $n_1 = 0$, then $n_2 = 0$ and $r_1 =$ abort, and by taking r_2 to be abort we have $r_2 \in [\![(x := e)^{(k)}]\!]^-(r_1)$. Assume $n_1 > 0$. Then $\mathrm{Pair2}(n_1, y_1, z_1)$, $\mathrm{EEval}_{e,\overrightarrow{x}}(y_1, w)$, $\mathrm{ChangeStore}_{\overrightarrow{x},x}(y_1, w, y_2)$ and $\mathrm{Pair2}(n_2, y_2, z_1)$ are true for some values for y_1, y_2, z_1, w. Now we have s, h such that $r_1 = (s, h)$. Let n be the value of w. Let $r_2 = (s[x := n], h)$. Then we have $[\![e]\!]_s = n$. Therefore $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$ and $r_2 \in [\![(x := e)^{(k)}]\!]^-(r_1)$.

Case 2. P is $P_1; P_2$.

Assume that $\mathrm{ExecU}_{P_1;P_2,\overrightarrow{x}}(k, n_1, n_2)$ is true and r_1 is given. We have z such that $\mathrm{ExecU}_{P_1,\overrightarrow{x}}(k, n_1, z) \land \mathrm{ExecU}_{P_2,\overrightarrow{x}}(k, z, n_2)$ is true. By induction hypothesis for P_1, we have r_0 such that $\mathrm{Result}_{\overrightarrow{x}}(z, r_0)$ and $r_0 \in [\![P_1^{(k)}]\!]^-(r_1)$ hold. By induction hypothesis for P_2, we have r_2 such that $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$ and $r_2 \in [\![P_2^{(k)}]\!]^-(r_0)$. Therefore $r_2 \in [\![(P_1; P_2)^{(k)}]\!]^-(r_1)$.

Case 3. P is if (b) then (P_1) else (P_2).

Assume that $\mathrm{ExecU}_{\mathrm{if}\,(b)\,\mathrm{then}\,(P_1)\,\mathrm{else}\,(P_2),\overrightarrow{x}}(k, n_1, n_2)$ is true and r_1 is given. If $n_1 = 0$, then $n_2 = 0$ and $r_1 =$ abort, and by taking r_2 to be abort we have $[\![(\mathrm{if}\,(b)\,\mathrm{then}\,(P_1)\,\mathrm{else}\,(P_2))^{(k)}]\!]^-(r_1) \ni r_2$. Assume $n_1 > 0$. We have x,y such that $\mathrm{Pair2}(n_1, x, y)$ is true. Let $r_1 = (s, h)$. Then $\mathrm{Storecode}_{\overrightarrow{x}}(x, s)$ holds.

Case 3.1. $[\![b]\!]_s = \mathrm{True}$. By Lemma 5.2.7 (2), $\mathrm{BEval}_{b,\vec{x}}(x)$ is true. Then $\mathrm{ExecU}_{P_1,\vec{x}}(k, n_1, n_2)$ is true. By induction hypothesis, we have r_2 such that $\mathrm{Result}_{\vec{x}}(n_2, r_2)$ and $[\![P_1^{(k)}]\!]^-(r_1) \ni r_2$. By definition, $[\![(\mathrm{if}\ (b)\ \mathrm{then}\ (P_1)\ \mathrm{else}\ (P_2))^{(k)}]\!]^-(r_1) \ni r_2$.

Case 3.2. $[\![b]\!]_s = \mathrm{False}$. In the same way as above, we can show this case.

Case 4. P is while (b) do (P_1).

Assume that $\mathrm{ExecU}_{\mathrm{while}\ (b)\ \mathrm{do}\ (P_1),\vec{x}}(k, n_1, n_2)$ is true and r_1 is given. If $n_1 = 0$, then $n_2 = 0$ and $r_1 = \mathrm{abort}$, and by taking r_2 to be abort we have $[\![(\mathrm{while}\ (b)\ \mathrm{do}\ (P_1))^{(k)}]\!]^-(r_1) \ni r_2$. Assume $n_1 > 0$. Let y_1, z_1 be such that $\mathrm{Pair2}(n_1, y_1, z_1)$ is true. We have $w = \langle w_1, \ldots, w_n\rangle$, $w_1 = n_1, w_n = n_2$ and $\mathrm{ExecU}_{P_1,\vec{x}}(k, w_i, w_{i+1})$ for all $0 < i < n$. We also have either $w_n = 0$ or $w_n > 0$ and the fact that $\mathrm{BEval}_{b,\vec{x}}(y_n)$ is true where $\mathrm{Pair2}(n_2, y_n, z_n)$ is true for some y_n, z_n. By repeatedly using induction hypothesis for P_1, we have $r'_1, \ldots, r'_n$ such that $r'_1 = r_1$, $\mathrm{Result}_{\vec{x}}(w_i, r'_i)$ for all $0 < i \le n$, and $[\![P_1^{(k)}]\!]^-(r'_i) \ni r'_{i+1}$ for all $0 < i < n$. By Lemma 5.2.7 (2), we also have either $r_n = \mathrm{abort}$ or $[\![b]\!]_{s_n} = \mathrm{False}$ and $r_n \neq \mathrm{abort}$ where $r_n = (s_n, h_n)$. Let $r_2 = r'_n$. By Proposition 4.2.4, $[\![\mathrm{while}\ (b)\ \mathrm{do}\ (P_1^{(k)})]\!]^-(r_1) \ni r_2$. Then by definition, $[\![(\mathrm{while}\ (b)\ \mathrm{do}\ (P_1))^{(k)}]\!]^-(r_1) \ni r_2$.

Case 5. P is skip.

Its proof is immediate.

Case 6. P is $x := \mathrm{cons}(e_1, e_2)$.

Assume that $\mathrm{ExecU}_{x:=\mathrm{cons}(e_1,e_2),\vec{x}}(k, n_1, n_2)$ is true and r_1 is given. If $n_1 = 0$, then $n_2 = 0$ and $r_1 = \mathrm{abort}$, and by taking r_2 to be abort we have $r_2 \in [\![(x := \mathrm{cons}(e_1, e_2))^{(k)}]\!]^-(r_1)$. Assume $n_1 > 0$. We have s, h such that $r_1 = (s, h)$. Let n be the value of w in the definition of $\mathrm{ExecU}_{x:=\mathrm{cons}(e_1,e_2),\vec{x}}(k, n_1, n_2)$. Let $r_2 = (s[x := n], h[n := [\![e_1]\!]_s, n+1 := [\![e_2]\!]_s])$. Then $n, n+1 \notin \mathrm{Dom}(h)$. Therefore $r_2 \in [\![(x := \mathrm{cons}(e_1, e_2))^{(k)}]\!]^-(r_1)$. We also have $\mathrm{Result}_{\vec{x}}(n_2, r_2)$.

Case 7. P is $x := [e]$.

Assume that $\mathrm{ExecU}_{x:=[e],\vec{x}}(k, n_1, n_2)$ is true and r_1 is given. If $n_1 = 0$, then $n_2 = 0$

and $r_1 = \text{abort}$, and by taking r_2 to be abort we have $r_2 \in [\![x := [e]]\!]^-(r_1)$. Assume $n_1 > 0$. We have s, h such that $r_1 = (s, h)$. Take r_2 such that either $r_2 = (s[x := h([\![e]\!]_s)], h)$ and $[\![e]\!]_s \in \text{Dom}(h)$ or $r_2 = \text{abort}$ and $[\![e_1]\!]_s \notin \text{Dom}(h)$. Then $r_2 \in [\![(x := [e])^{(k)}]\!]^-(r_1)$. We also have $\text{Result}_{\vec{x}}(n_2, r_2)$.

Case 8. P is $[e_1] := e_2$.

Assume that $\text{ExecU}_{[e_1]:=e_2,\vec{x}}(k, n_1, n_2)$ is true and r_1 is given. If $n_1 = 0$, then $n_2 = 0$ and $r_1 = \text{abort}$, and by taking r_2 to be abort we have $r_2 \in [\![[e_1] := e_2]\!]^-(r_1)$. Assume $n_1 > 0$. We have s, h such that $r_1 = (s, h)$. Take r_2 such that either $[\![e_1]\!]_s \in \text{Dom}(h)$ and $r_2 = (s, h[[\![e_1]\!]_s := [\![e_2]\!]_s])$ or $[\![e_1]\!]_s \notin \text{Dom}(h)$ and $r_2 = \text{abort}$. Then $r_2 \in [\![([e_1] := e_2)^{(k)}]\!]^-(r_1)$. We also have $\text{Result}_{\vec{x}}(n_2, r_2)$.

Case 9. P is $\text{dispose}(e)$.

Assume that $\text{ExecU}_{\text{dispose}(e),\vec{x}}(k, n_1, n_2)$ is true and r_1 is given. If $n_1 = 0$, then $n_2 = 0$ and $r_1 = \text{abort}$, and by taking r_2 to be abort we have $r_2 \in [\![\text{dispose}(e)]\!]^-(r_1)$. Assume $n_1 > 0$. We have s, h such that $r_1 = (s, h)$. Take r_2 such that either $r_2 = (s, h|_{\text{Dom}(h)-\{[\![e]\!]_s\}})$ and $[\![e]\!]_s \in \text{Dom}(h)$ or $r_2 = \text{abort}$ and $[\![e_1]\!]_s \notin \text{Dom}(h)$. Then $r_2 \in [\![(\text{dispose}(e))^{(k)}]\!]^-(r_1)$. We also have $\text{Result}_{\vec{x}}(n_2, r_2)$.

Case 10. P is R_i. We consider cases according to k.

Case 10.1. $k = 0$.

Assume $\text{ExecU}_{R_i,\vec{x}}(0, n_1, n_2)$ is true and r_1 is given. By definition, $n_1 = n_2 = 0$ and $r_1 = \text{abort}$. Let r_2 be abort. Then the claim holds.

Case 10.2. $k = k' + 1$.

By definition $\text{ExecU}_{R_i,\vec{x}}(k' + 1, n_1, n_2) = \text{ExecU}_{Q_i,\vec{x}}(k', n_1, n_2)$ and $Q_i^{(k')} = R_i^{(k'+1)}$. By induction hypothesis for k', the claim holds for $\text{ExecU}_{Q_i,\vec{x}}(k', n_1, n_2)$ and $[\![Q_i^{(k')}]\!]^-(r_1) \ni r_2$. Hence the claim holds for $\text{ExecU}_{R_i,\vec{x}}(k' + 1, n_1, n_2)$ and $[\![R_i^{(k'+1)}]\!]^-(r_1) \ni r_2$.

(2) We will prove it by induction on (k, P). We will consider the cases of P.

Case 1. P is $x := e$.

Assume the conditions. We will show that $\mathrm{ExecU}_{x:=e,\vec{x}}(k, n_1, n_2)$ is true. If $r_1 =$ abort, then $r_2 =$ abort and we have $n_1 = n_2 = 0$, and hence $\mathrm{ExecU}_{x:=e,\vec{x}}(k, n_1, n_2)$ is true. Now assume $r_1 = (s, h)$. We have some n_1, n_2 such that $\mathrm{Result}_{\vec{x}}(n_1, r_1)$ and $\mathrm{Result}_{\vec{x}}(n_2, r_2)$ hold. Then $n_1 > 0$. We have y_1, z_1 such that $\mathrm{Pair2}(n_1, y_1, z_1)$ is true and $\mathrm{Storecode}_{\vec{x}}(y_1, s)$ and $\mathrm{Heapcode}(z_1, h)$ hold. We also have $\mathrm{EEval}_{e,\vec{x}}(y_1, n)$.

Then $r_2 = (s_2, h)$ where $s_2 = (s[x := n])$. Then we have y_2, z_2 such that $\mathrm{Storecode}(y_2, s_2)$ and $\mathrm{Heapcode}(z_2, h)$ hold. Then $\mathrm{ChangeStore}_{\vec{x},x}(y_1, n, y_2)$ is true and $z_1 = z_2$. Then by definition, $\mathrm{ExecU}_{x:=e,\vec{x}}(k, n_1, n_2)$ is true.

Case 2. P is $P_1; P_2$.

Assume the conditions. We will show that $\mathrm{ExecU}_{P,\vec{x}}(k, n_1, n_2)$ is true. We have $[\![(P_1; P_2)^{(k)}]\!]^-(r_1) \ni r_2$. By definition, we have r_0 such that $r_0 \in [\![P_1^{(k)}]\!]^-(r_1)$ and $r_2 \in [\![P_2^{(k)}]\!]^-(r_0)$. Suppose z is such that $\mathrm{Result}_{\vec{x}}(z, r_0)$ holds. By induction hypothesis for P_1, $\mathrm{ExecU}_{P_1,\vec{x}}(k, n_1, z)$ is true. By induction hypothesis for P_2, $\mathrm{ExecU}_{P_2,\vec{x}}(k, z, n_2)$ is true. Hence by definition $\mathrm{ExecU}_{P,\vec{x}}(k, n_1, n_2)$ is true.

Case 3. P is if (b) then (P_1) else (P_2).

Assume the conditions. We will show that $\mathrm{ExecU}_{P,\vec{x}}(k, n_1, n_2)$ is true. We have $[\![P^{(k)}]\!]^-(r_1) \ni r_2$. If $r_1 =$ abort, then $r_2 =$ abort and we have $n_1 = n_2 = 0$, and hence $\mathrm{Exec}_{P,\vec{x}}(k, n_1, n_2)$ is true. Assume $r_1 = (s, h)$. Then $n_1 > 0$. We have y_1,z_1 such that $\mathrm{Pair2}(n_1, y_1, z_1)$ is true.

Case 3.1. $\mathrm{BEval}_{b,\vec{x}}(y_1)$ is true. By Lemma 5.2.7 (2), $\mathrm{Storecode}_{\vec{x}}(y_1, s)$ and $[\![b]\!]_s =$ True hold. Then by definition, $[\![P_1^{(k)}]\!]^-(r_1) \ni r_2$. By induction hypothesis, $\mathrm{ExecU}_{P_1,\vec{x}}(k, n_1, n_2)$ is true. Then by definition, $\mathrm{ExecU}_{P,\vec{x}}(k, n_1, n_2)$ is true.

Case 3.2. $\mathrm{BEval}_{b,\vec{x}}(y_1)$ is false. In the same way as above we can show the claim.

Case 4. P is while (b) do (P_1).

Assume the conditions. We have $[\![(\mathrm{while}\ (b)(P_1))^{(k)}]\!]^-(r_1) \ni r_2$. If $r_1 =$ abort, then $r_2 =$ abort and we have $n_1 = n_2 = 0$. Hence $\mathrm{ExecU}_{\mathrm{while}\ (b)\ \mathrm{do}\ (P_1),\vec{x}}(k, n_1, n_2)$ is true. Now assume $r_1 = (s, h)$. By Proposition 4.2.4, we have $(s_1, h_1), \ldots, (s_{m-1}, h_{m-1}), r_m$ such that $(s, h) = (s_1, h_1)$, for all $i = 1, \ldots, m -$

2, $[\![P_1^{(k)}]\!]^-((s_i,h_i)) \ni (s_{i+1},h_{i+1})$, $[\![b]\!]_{s_i} = \text{True}$, $[\![P_1^{(k)}]\!]^-((s_{m-1},h_{m-1})) \ni r_m$, either $[\![b]\!]_{s_{m-1}} = \text{True}$ and $r_m = \text{abort}$ or $r_m = (s_m,h_m)$ and $[\![b]\!]_{s_m} = \text{False}$ for some s_m, h_m.

Then we have $z_1,\ldots,z_m$ such that for all $i = 1,\ldots,m-1$, $\text{Result}_{\overrightarrow{x}}(z_i,(s_i,h_i))$ holds and either $z_m = 0$ or $\text{Result}_{\overrightarrow{x}}(z_m,(s_m,h_m))$ holds. Then $z_1 = n_1$ and $z_m = n_2$. We also have $y_1,\ldots,y_m,y'_1,\ldots,y'_m$ such that for all $i = 1,\ldots,m-1$, $\text{BEval}_{b,\overrightarrow{x}}(y_i)$ is true where $\text{Pair2}(z_i,y_i,y'_i)$ is true, and either $z_m = 0$ or $\text{Pair2}(z_m,y_m,y'_m)$ is true and $\text{BEval}_{b,\overrightarrow{x}}(y_m)$ is false. For all $i = 1,\ldots,m-1$, by induction hypothesis, $\text{ExecU}_{P_1,\overrightarrow{x}}(k,z_i,z_{i+1})$ is true. Then by definition, $\text{ExecU}_{\text{while }(b)\text{ do }(P_1),\overrightarrow{x}}(k,n_1,n_2)$ is true.

Case 5. P is skip.

Its proof is immediate.

Case 6. P is $x := \text{cons}(e_1,e_2)$.

Assume the conditions. We will show that $\text{ExecU}_{x:=\text{cons}(e_1,e_2),\overrightarrow{x}}(k,n_1,n_2)$ is true. We have $r_2 \in [\![(x := \text{cons}(e_1,e_2))^{(k)}]\!]^-(r_1)$. If $r_1 = \text{abort}$, then $r_2 = \text{abort}$ and we have $n_1 = n_2 = 0$, and hence $\text{ExecU}_{x:=\text{cons}(e_1,e_2),\overrightarrow{x}}(k,n_1,n_2)$ is true. Now assume $r_1 = (s,h)$. Then $r_2 = (s_2,h_2)$ where $s_2 = (s[x := n])$, $h_2 = h[n := [\![e_1]\!]_s, n+1 := [\![e_2]\!]_s]$ and $n, n+1 \notin \text{Dom}(h)$. Then $n_1 > 0$. We also have y_1,z_1,y_2,z_2 such that $\text{Pair2}(n_1,y_1,z_1)$ and $\text{Pair2}(n_2,y_2,z_2)$ are true. Then $\text{Storecode}_{\overrightarrow{x}}(y_1,s)$, $\text{Heapcode}(z_1,h)$, $\text{Storecode}_{\overrightarrow{x}}(y_2,s_2)$ and $\text{Heapcode}(z_2,h_2)$ hold. Let $w = n$, $w_1 = [\![e_1]\!]_s$ and $w_2 = [\![e_2]\!]_s$. Then by Lemma 5.2.7 (1), $\text{EEval}_{e_1,\overrightarrow{x}}(y_1,w_1)$, $\text{EEval}_{e_2,\overrightarrow{x}}(y_1,w_2)$ and $\text{New2}(w,z_1)$ are true. Then $\text{ChangeStore}_{\overrightarrow{x},x}(y_1,w,y_2)$, $\forall xy(x \neq w \wedge x \neq w+1 \rightarrow (\text{Lookup}(z_1,x,y) \leftrightarrow \text{Lookup}(z_2,x,y)))$ and $\text{Lookup}(z_2,w,w_1) \wedge \text{Lookup}(z_2,w+1,w_2)$ are true. Then by definition, $\text{ExecU}_{x:=\text{cons}(e_1,e_2),\overrightarrow{x}}(k,n_1,n_2)$ is true.

Case 7. P is $x := [e]$.

Assume the conditions. We will show that $\text{ExecU}_{x:=[e],\overrightarrow{x}}(k,n_1,n_2)$ is true. We have $r_2 \in [\![(x := [e])^{(k)}]\!]^-(r_1)$. If $r_1 = \text{abort}$, then $r_2 = \text{abort}$ and we have $n_1 = n_2 = 0$, and hence $\text{ExecU}_{x:=[e],\overrightarrow{x}}(k,n_1,n_2)$ is true. Now assume $r_1 = (s,h)$. Then $n_1 > 0$. We have y_1,z_1 such that $\text{Pair2}(n_1,y_1,z_1)$ is true and $\text{Storecode}_{\overrightarrow{x}}(y_1,s)$ and $\text{Heapcode}(z_1,h)$ hold. Let w be $[\![e]\!]_s$.

Assume that $\mathrm{Domain}(w, z_1)$ is true. By Lemma 5.2.7 (1), $\mathrm{EEval}_{e,\vec{x}}(y_1, w)$ is true. Then $[\![e]\!]_s \in \mathrm{Dom}(h)$. Then $r_2 = (s_2, h_2)$ where $h(w) = w_1$, $s_2 = (s[x := w_1])$ and $h_2 = h$. Then we have y_2 such that $\mathrm{Pair2}(n_2, y_2, z_1)$ is true and $\mathrm{Storecode}_{\vec{x}}(y_2, s_2)$ and $\mathrm{Heapcode}(z_1, h_2)$ hold. Then $\mathrm{Lookup}(z_1, w, w_1)$ and $\mathrm{ChangeStore}_{\vec{x},x}(y_1, w_1, y_2)$ are true. Now assume that $\neg\mathrm{Domain}(w, z_1)$ is true. Then $[\![e]\!]_s \notin \mathrm{Dom}(h)$. Then $r_2 = \mathrm{abort}$ and hence $n_2 = 0$. Then by definition, $\mathrm{ExecU}_{x:=[e],\vec{x}}(k, n_1, n_2)$ is true in both cases.

Case 8. P is $[e_1] := e_2$.

Assume the conditions. We will show that $\mathrm{ExecU}_{[e_1]:=e_2,\vec{x}}(k, n_1, n_2)$ is true. We have $r_2 \in [\![([e_1] := e_2)^{(k)}]\!]^-(r_1)$. If $r_1 = \mathrm{abort}$, then $r_2 = \mathrm{abort}$ and we have $n_1 = n_2 = 0$, and hence $\mathrm{ExecU}_{[e_1]:=e_2,\vec{x}}(k, n_1, n_2)$ is true. Now assume $r_1 = (s, h)$. Then $n_1 > 0$. We have y_1, z_1 such that $\mathrm{Pair2}(n_1, y_1, z_1)$ is true and $\mathrm{Storecode}_{\vec{x}}(y_1, s)$ and $\mathrm{Heapcode}(z_1, h)$ hold. Let w be $[\![e_1]\!]_s$ and w_1 be $[\![e_2]\!]_s$. Assume that $\mathrm{Domain}(w, z_1)$ is true. By Lemma 5.2.7 (1), $\mathrm{EEval}_{e_1,\vec{x}}(y_1, w)$ and $\mathrm{EEval}_{e_2,\vec{x}}(y_1, w_1)$ are true. Then we have $w \in \mathrm{Dom}(h)$. Then $r_2 = (s_2, h_2)$ where $s_2 = s$ and $h_2 = h[w := w_1]$. Then we have z_2 such that $\mathrm{Pair2}(n_2, y_1, z_2)$ is true and $\mathrm{Storecode}_{\vec{x}}(y_1, s_2)$ and $\mathrm{Heapcode}(z_2, h_2)$ hold. Then $\mathrm{ChangeHeap}(z_1, w, w_1, z_2)$ is true. Now assume that $\neg\mathrm{Domain}(w, z_1)$ is true. Then $[\![e]\!]_s \notin \mathrm{Dom}(h)$. Then $r_2 = \mathrm{abort}$ and hence $n_2 = 0$. Then by definition, $\mathrm{ExecU}_{[e_1]:=e_2,\vec{x}}(k, n_1, n_2)$ is true in both cases.

Case 9. P is $\mathrm{dispose}(e)$.

Assume the conditions. We will show that $\mathrm{ExecU}_{\mathrm{dispose}(e),\vec{x}}(k, n_1, n_2)$ is true. We have $r_2 \in [\![(\mathrm{dispose}(e))^{(k)}]\!]^-(r_1)$. If $r_1 = \mathrm{abort}$, then $r_2 = \mathrm{abort}$ and we have $n_1 = n_2 = 0$, and hence $\mathrm{ExecU}_{\mathrm{dispose}(e),\vec{x}}(k, n_1, n_2)$ is true. Now assume $r_1 = (s, h)$. Then $n_1 > 0$. We have y_1, z_1 such that $\mathrm{Pair2}(n_1, y_1, z_1)$ is true and $\mathrm{Storecode}_{\vec{x}}(y_1, s)$ and $\mathrm{Heapcode}(z_1, h)$ hold. Let w be $[\![e]\!]_s$. By Lemma 5.2.7 (1), $\mathrm{EEval}_{e,\vec{x}}(y_1, w)$ is true. Assume that $\mathrm{Domain}(w, z_1)$ is true. Then we have $[\![e]\!]_s \in \mathrm{Dom}(h)$. Then $r_2 = (s_2, h_2)$ where $s_2 = s$ and $h_2 = h|_{\mathrm{Dom}(h)-\{[\![e]\!]_s\}}$. Then we have z_2 such that $\mathrm{Pair2}(n_2, y_1, z_2)$ is true and $\mathrm{Storecode}_{\vec{x}}(y_1, s_2)$ and $\mathrm{Heapcode}(z_2, h_2)$ hold. Then $\forall xy(\mathrm{Lookup}(z_1, x, y) \wedge x \neq w \leftrightarrow \mathrm{Lookup}(z_2, x, y))$ is true. Now assume that $\neg\mathrm{Domain}(w, z_1)$ is true. Then $[\![e]\!]_s \notin \mathrm{Dom}(h)$. Then $r_2 = \mathrm{abort}$ and hence $n_2 = 0$. Then by definition, $\mathrm{ExecU}_{\mathrm{dispose}(e),\vec{x}}(k, n_1, n_2)$ is true in both cases.

Case 10. P is R_i. We consider cases according to k.

Case 10.1. $k = 0$.

Assume that the conditions. By Proposition 4.2.5 $[\![R_i^{(0)}]\!]^-(\mathrm{abort}) \ni \mathrm{abort}$ and for all s, h, $[\![R_i^{(0)}]\!]^-((s, h)) = \emptyset$. Hence we have $r_1 = r_2 = \mathrm{abort}$. Since $\mathrm{Result}_{\overrightarrow{x}}(n_1, r_1)$ and $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$ hold, by definition $n_1 = n_2 = 0$. Then by definition $\mathrm{ExecU}_{R_i, \overrightarrow{x}}(0, n_1, n_2)$ is true.

Case 10.2. $k = k' + 1$.

This case is proved in a similar way to that in (1). $\square$

The next lemma shows that the *pure* formula $\mathrm{Exec}_{P, \overrightarrow{x}}(n_1, n_2)$ actually have the meaning we explained above.

Lemma 5.2.9 *(1) If $\mathrm{Exec}_{P, \overrightarrow{x}}(n_1, n_2)$ is true, then for all r_1 such that $\mathrm{Result}_{\overrightarrow{x}}(n_1, r_1)$, we have r_2 such that $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$ and $[\![P]\!](r_1) \ni r_2$.*

(2) If $[\![P]\!](r_1) \ni r_2$, $\mathrm{Result}_{\overrightarrow{x}}(n_1, r_1)$, and $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$ hold, then $\mathrm{Exec}_{P, \overrightarrow{x}}(n_1, n_2)$ is true.

Proof. (1) By using Lemma 5.2.8 (1), we can prove the claim in a similar way to (2).

(2) Assume $[\![P]\!](r_1) \ni r_2$, $\mathrm{Result}_{\overrightarrow{x}}(n_1, r_1)$, and $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$. Then we have k such that $[\![P^{(k)}]\!]^-(r_1) \ni r_2$. From Lemma 5.2.8 (2), $\mathrm{ExecU}_{P, \overrightarrow{x}}(k, n_1, n_2)$ is true. Hence $\mathrm{ExecU}_{P, \overrightarrow{x}}(k, n_1, n_2)$ is true. $\square$

5.2.5 Weakest Precondition

We define the weakest precondition for a program and a postcondition. We also define a formula $\mathrm{W}_{P,A}(\overrightarrow{x})$ and show that it is the weakest precondition of the program P and the postcondition A.

Definition 5.2.10 *For a program P and an assertion A, the weakest precondition for P and A under the standard interpretation is defined as the set $\{ (s, h) \mid \forall r([\![P]\!]((s, h)) \ni r \to r \neq abort \wedge [\![A]\!]_r = True) \}$.*

Since we have defined Exec and have shown Lemma 5.2.9 for our extended programming language with procedure calls, we can show the existence of the assertion of the weakest precondition in the same way as [19].

Definition 5.2.11 *We define the formula $W_{P,A}(\overrightarrow{x})$ for the program P and the assertion A. We fix some sequence $\overrightarrow{x}$ of the variables that includes $EFV(P) \cup FV(A)$.*

$$\begin{aligned} W_{P,A}(\overrightarrow{x}) = \forall xyzw(&Store_{\overrightarrow{x}}(x) \wedge Heap(y) \wedge Pair2(z,x,y) \wedge Exec_{P,\overrightarrow{x}}(z,w) \to \\ & w > 0 \wedge \exists y_1 z_1(Pair2(w,y_1,z_1) \wedge Eval_{A,\overrightarrow{x}}(y_1,z_1))). \end{aligned}$$

$\mathrm{W}_{P,A}(\overrightarrow{x})$ means the weakest precondition for P and A. That is, $\mathrm{W}_{P,A}(\overrightarrow{x})$ gives the weakest assertion W such that $\{W\}P\{A\}$ is true. Note that all the free variables in $\mathrm{W}_{P,A}(\overrightarrow{x})$ are $\overrightarrow{x}$ and they appear only in $\mathrm{Store}_{\overrightarrow{x}}(x)$.

The next lemma says that $\mathrm{W}_{P,A}(\overrightarrow{x})$ indeed describes the weakest precondition for P and A.

Lemma 5.2.12 *(1) $\{W_{P,A}(\overrightarrow{x})\}P\{A\}$ is true.*

(2) If $[\![P]\!]((s,h)) \ni r$ implies $r \neq$ abort and $[\![A]\!]_r = $ True for all r, then $[\![W_{P,A}(\overrightarrow{x})]\!]_{(s,h)} = $ True.

Proof. (1) Assume $[\![\mathrm{W}_{P,A}(\overrightarrow{x})]\!]_{(s,h)} = \mathrm{True}$ and $[\![P]\!]((s,h)) \ni r$. We will show $r \neq \mathrm{abort}$ and $[\![A]\!]_r = \mathrm{True}$. We have n_1,n_2,n and m such that $\mathrm{Result}_{\overrightarrow{x}}(n_1,(s,h))$ and $\mathrm{Result}_{\overrightarrow{x}}(n_2,r)$ hold and $\mathrm{Pair2}(n_1,n,m)$ is true. We have $[\![\mathrm{Store}_{\overrightarrow{x}}(n)]\!]_{(s,h)}$ and $[\![\mathrm{Heap}(m)]\!]_{(s,h)}$ are true and $\mathrm{Storecode}_{\overrightarrow{x}}(n,s)$ and $\mathrm{Heapcode}(m,h)$ hold. By Lemma 5.2.9 (2), $\mathrm{Exec}_{P,\overrightarrow{x}}(n_1,n_2)$ is true.

By letting $x = n, y = m, z = n_1, w = n_2$ in the definition of $\mathrm{W}_{P,A}(\overrightarrow{x})$, from $[\![\mathrm{W}_{P,A}(\overrightarrow{x})]\!]_{(s,h)} = \mathrm{True}$, we have $n_2 > 0 \wedge \exists y_1 z_1(\mathrm{Pair2}(n_2,y_1,z_1) \wedge \mathrm{Eval}_{A,\overrightarrow{x}}(y_1,z_1))$. By $n_2 > 0$, $r \neq \mathrm{abort}$. Let $r = (s_1,h_1)$. We have n', m' such that $\mathrm{Pair2}(n_2,n',m')$ and $\mathrm{Eval}_{A,\overrightarrow{x}}(n',m')$ are true. By Lemma 5.2.7 (4), we have s_1', h_1' such that $\mathrm{Storecode}_{\overrightarrow{x}}(n',s_1') \wedge \mathrm{Heapcode}(m',h_1') \wedge [\![A]\!]_{(s_1',h_1')}$ holds. Since $\mathrm{Storecode}_{\overrightarrow{x}}(n',s_1)$ and $\mathrm{Heapcode}(m',h_1)$ hold, we have $s_1' =_{\overrightarrow{x}} s_1$ and $h_1' = h_1$. Hence $[\![A]\!]_{(s_1,h_1)} = \mathrm{True}$, that is, $[\![A]\!]_r = \mathrm{True}$.

(2) Assume that for all r, $[\![P]\!]((s,h)) \ni r$ implies $r \neq \text{abort} \wedge [\![A]\!]_r = \text{True}$. We will show $[\![\mathrm{W}_{P,A}(\overrightarrow{x})]\!]_{(s,h)} = \text{True}$. Fix n, m, n_1, n_2 and assume $\mathrm{Store}_{\overrightarrow{x}}(n)$, $\mathrm{Heap}(m)$, $\mathrm{Pair2}(n_1, n, m)$, and $\mathrm{Exec}_{P,\overrightarrow{x}}(n_1, n_2)$ are true at (s,h). We will show that $n_2 > 0$ and $\exists y_1 z_1(\mathrm{Pair2}(n_2, y_1, z_1) \wedge \mathrm{Eval}_{A,\overrightarrow{x}}(y_1, z_1))$ is true.

We have $\mathrm{Result}_{\overrightarrow{x}}(n_1, (s,h))$. By Lemma 5.2.9 (1) with $\mathrm{Exec}_{P,\overrightarrow{x}}(n_1, n_2)$, we have r_2 such that $[\![P]\!]((s,h)) \ni r_2$ and $\mathrm{Result}_{\overrightarrow{x}}(n_2, r_2)$. By the assumption, $[\![A]\!]_{r_2} = \text{True}$. Let r_2 be (s_2, h_2). We have n', m' such that $\mathrm{Pair2}(n_2, n', m')$ is true. Then $\mathrm{Storecode}_{\overrightarrow{x}}(n', s_2)$ and $\mathrm{Heapcode}(m', h_2)$ hold. Since the right-hand side of Lemma 5.2.7 (4) holds by letting $s = s_2$ and $h = h_2$, we have $\mathrm{Eval}_{A,\overrightarrow{x}}(n', m')$. Hence $\exists y_1 z_1(\mathrm{Pair2}(n_2, y_1, z_1) \wedge \mathrm{Eval}_{A,\overrightarrow{x}}(y_1, z_1))$ is true by taking $y_1 = n'$ and $z_1 = m'$.

Therefore, $\forall xyzw'(\mathrm{Store}_{\overrightarrow{x}}(x) \wedge \mathrm{Heap}(y) \wedge \mathrm{Pair2}(z, x, y) \wedge \mathrm{Exec}_{P,\overrightarrow{x}}(z, w') \to w' > 0 \wedge \exists y_1 z_1(\mathrm{Pair2}(w', y_1, z_1) \wedge \mathrm{Eval}_{A,\overrightarrow{x}}(y_1, z_1)))$ is true at (s,h), that is, $[\![\mathrm{W}_{P,A}(\overrightarrow{x})]\!]_{(s,h)} = \text{True}$. □

We present the main theorem about expressiveness.

Theorem 5.2.13 (Expressiveness) *Our assertion language is expressive for our programming language under the standard interpretation. Namely for every program P and assertion A, there is a formula W such that $[\![W]\!]_{(s,h)}$ is true if and only if (s,h) is in the weakest precondition for P and A under the standard interpretation.*

Proof. Since Lemma 5.2.12 (1) and (2) show $\mathrm{W}_{P,A}(\overrightarrow{x})$ defines the weakest precondition for P and A under the standard interpretation, the weakest precondition is definable in our language. □

5.3 Completeness

This section is the most important section of this paper. It shows that our system is complete. Although the proof technique here is inspired from [3], we introduce some important concepts to show the completeness of our system. We also define the strongest postcondition in the same way as the weakest precondition in order to follow a similar story of proofs in [3].

For the form of $\Gamma \vdash \{A\}P\{B\}$ where Γ is empty, we will write $\vdash \{A\}P\{B\}$.

5.3.1 Strongest Postcondition

$\{A\}P\{\text{True}\}$ is true when P does not abort at a state for which A is true. We call $\{A\}P\{\text{True}\}$ the abort-free condition for A and P.

The set S of states is called the strongest postcondition for A and P if
(1) For all r, r', $[\![A]\!]_r = \text{True}$ and $[\![P]\!](r) \ni r'$ implies $r' \neq \text{abort}$ and $r' \in S$.
(2) For all S', we have $S \subseteq S'$ if for all r, r', $[\![A]\!]_r = \text{True}$ and $[\![P]\!](r) \ni r'$ implies $r' \neq \text{abort}$ and $r' \in S'$.

Note that the strongest postcondition S for A and P exists if P does not abort at any state that satisfies A. Since we have defined Exec and have shown Lemma 5.2.9 for our extended programming language with procedure calls, we can define the assertion that describes the strongest postcondition of A and P.

Definition 5.3.1 *We define the pure formula $S_{A,P}(\overrightarrow{x})$ for the assertion A and the program P. We fix some sequence $\overrightarrow{x}$ of the variables that includes $EFV(P) \cup FV(A)$.*

$$S_{A,P}(\overrightarrow{x}) = \exists xyzw(Eval_{A,\overrightarrow{x}}(x,y) \wedge Pair2(z,x,y) \wedge Exec_{P,\overrightarrow{x}}(z,w) \wedge \\ \exists y_1 z_1(Pair2(w,y_1,z_1) \wedge Store_{\overrightarrow{x}}(y_1) \wedge Heap(z_1))).$$

$S_{A,P}(\overrightarrow{x})$ describes the strongest postcondition for A and P. That is, $S_{A,P}(\overrightarrow{x})$ gives the strongest assertion B such that $\{A\}P\{B\}$ is true, under the condition that P does not abort at any state that satisfies A. Note that all the free variables in $S_{A,P}(\overrightarrow{x})$ are $\overrightarrow{x}$

and they appear only in $\mathrm{Store}_{\vec{x}}(x)$.

Lemma 5.3.2 *(1) If $\{A\}P\{True\}$ is true then $\{A\}P\{S_{A,P}(\vec{x})\}$ is true.*

(2) If $[\![S_{A,P}(\vec{x})]\!]_{(s',h')}$ is true then there exists s,h such that $[\![A]\!]_{(s,h)}$ is true and $[\![P]\!]((s,h)) \ni (s',h')$.

Proof. (1) Assume that $\{A\}P\{\mathrm{True}\}$ is true. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. Then $[\![P]\!]((s,h)) \not\ni \mathrm{abort}$. Assume $[\![P]\!]((s,h)) \ni (s',h')$. Let $\vec{x} = x_0,\ldots,x_n$. Let n_1 be $\langle s(x_0),\ldots,s(x_n)\rangle$ and m_1 be $\langle (l_0,h(l_0)),\ldots,(l_m,h(l_m))\rangle$ where $\mathrm{Dom}(h) = \{\, l_i \mid i \le m \,\}$. Let n_2 be $\langle s'(x_0),\ldots,s'(x_n)\rangle$, m_2 be $\langle (l'_0,h'(l'_0)),\ldots,(l'_{m'},h'(l'_{m'}))\rangle$ where $\mathrm{Dom}(h') = \{\, l'_i \mid i \le m' \,\}$. Let p_1 be $(n_1,m_1)+1$ and p_2 be $(n_2,m_2)+1$. Then $\mathrm{Pair2}(p_1,n_1,m_1)$ and $\mathrm{Pair2}(p_2,n_2,m_2)$ are true. Then $\mathrm{Storecode}_{\vec{x}}(n_1,s)$, $\mathrm{Heapcode}(m_1,h)$, $\mathrm{Storecode}_{\vec{x}}(n_2,s')$ and $\mathrm{Heapcode}(m_2,h')$ hold. Then we have $\mathrm{Result}_{\vec{x}}(p_1,(s,h))$ and $\mathrm{Result}_{\vec{x}}(p_2,(s',h'))$. Then $\exists sh(\mathrm{Storecode}_{\vec{x}}(n_1,s) \wedge \mathrm{Heapcode}(m_1,h) \wedge [\![A]\!]_{(s,h)} = \mathrm{True})$ holds. By Lemma 5.2.7 (4), $\mathrm{Eval}_{A,\vec{x}}(n_1,m_1)$ is true. By Lemma 5.2.9 (2), $\mathrm{Exec}_{P,\vec{x}}(p_1,p_2)$ is true. By definition, we also have $[\![\mathrm{Store}_{\vec{x}}(n_2) \wedge \mathrm{Heap}(m_2)]\!]_{(s',h')}$ is true. By taking y,z,w,w_1,y_1,z_1 to be n_1,m_1,p_1,p_2,n_2,m_2, $[\![\exists yzww_1y_1z_1(\mathrm{Eval}_{A,\vec{x}}(y,z) \wedge \mathrm{Pair2}(w,y,z) \wedge \mathrm{Exec}_{P,\vec{x}}(w,w_1) \wedge \mathrm{Pair2}(w_1,y_1,z_1) \wedge \mathrm{Store}_{\vec{x}}(y_1) \wedge \mathrm{Heap}(z_1))]\!]_{(s',h')}$ is true. Then by definition, $[\![S_{A,P}(\vec{x})]\!]_{(s',h')} = \mathrm{True}$. Therefore, $\{A\}P\{S_{A,P}(\vec{x})\}$ is true.

(2) Assume that $[\![S_{A,P}(\vec{x})]\!]_{(s',h')}$ is true. By definition, we have n_1,m_1,p_1,p_2,n_2,m_2 such that $\mathrm{Eval}_{A,\vec{x}}(n_1,m_1)$, $\mathrm{Pair2}(p_1,n_1,m_1)$, $\mathrm{Exec}_{P,\vec{x}}(p_1,p_2)$, $\mathrm{Pair2}(p_2,n_2,m_2)$, $[\![\mathrm{Store}_{\vec{x}}(n_2)]\!]_{(s',h')} = \mathrm{True}$ and $[\![\mathrm{Heap}(m_2)]\!]_{(s',h')} = \mathrm{True}$ hold. Then we have $\mathrm{Storecode}_{\vec{x}}(n_2,s')$ and $\mathrm{Heapcode}(m_2,h')$. By Lemma 5.2.7 (4) with $\mathrm{Eval}_{A,\vec{x}}(n_1,m_1)$, we have s_1,h such that $\mathrm{Storecode}_{\vec{x}}(n_1,s_1)$, $\mathrm{Heapcode}(m_1,h)$, and $[\![A]\!]_{(s_1,h)} = \mathrm{True}$. Then we have $\mathrm{Result}_{\vec{x}}(p_1,(s_1,h))$. Since we have $\mathrm{Exec}_{P,\vec{x}}(p_1,p_2)$, by Lemma 5.2.9 (1), we have r_2 such that $\mathrm{Result}_{\vec{x}}(p_2,r_2)$ and $[\![P]\!]((s_1,h)) \ni r_2$. Since $p_2 > 0$, we have $r_2 \ne \mathrm{abort}$. Take s_2,h' such that $r_2 = (s_2,h')$. We define s to be $[s_1,s',\vec{x}]$. Since $s =_{\mathrm{EFV}(P)} s_1$, by Lemma 4.2.16 (2), $[\![P]\!]((s,h)) \ni ([s_2,s,\mathrm{EFV}(P)],h')$.

We will show $s' = [s_2,s,\mathrm{EFV}(P)]$. We have $s_2 =_{\vec{x}} s'$ since $\mathrm{Storecode}(n_2,s_2)$ and $\mathrm{Storecode}(n_2,s')$. Hence $s' =_{\mathrm{EFV}(P)} s_2$. By the definition of s, we have $s' =_{(\vec{x})^c} s$. Since $s =_{\vec{x}} s_1$ by the definition of s, $s_1 =_{\mathrm{EFV}(P)^c} s_2$ by Lemma 4.2.16 (3), and $s_2 =_{\vec{x}} s'$, we have $s =_{\overrightarrow{x}-\mathrm{EFV}(P)} s'$. Hence $s =_{\mathrm{EFV}(P)^c} s'$. Therefore $s' = [s_2, s, \mathrm{EFV}(P)]$.

Hence $[\![P]\!]((s,h)) \ni (s',h')$. We also have $[\![A]\!]_{(s,h)}$ is true since $s =_{\overrightarrow{x}} s_1$. $\square$

Remark. The following is shown by Lemma 5.3.2 (2): if $\{A\}\{P\}\{B\}$ is true, then $S_{A,P}(\overrightarrow{x}) \to B$ is true.

5.3.2 Auxiliary Lemmas

Lemma 5.3.3 *$A \to \exists x(Heap(x) \wedge HEval_A(x))$ is true.*

Proof. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. Let m be $\langle (k_0, l_0), \ldots, (k_n, l_n)\rangle$ where $h(k_i) = l_i$ for $i = 0, \ldots, n$ and $\mathrm{Dom}(h) = \{\, k_i \mid i = 0, \ldots, n \,\}$ and $0 < k_0 < \ldots < k_n$. Then $\mathrm{Heapcode}(m,h)$ holds. Then $[\![\mathrm{Heap}(m)]\!]_{(s,h)} = \mathrm{True}$. Then by Lemma 5.2.7 (3), $[\![\mathrm{HEval}_A(m)]\!]_{(s,h)} = \mathrm{True}$. By taking x to be m, $\exists x(\mathrm{Heap}(x) \wedge \mathrm{HEval}_A(x))$ is true. $\square$

Lemma 5.3.4 *If $\{A\}P\{B\}$ is true, then $A \to W_{P,B}(\overrightarrow{x})$ is true.*

Proof. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. We will show $[\![\mathrm{W}_{P,B}(\overrightarrow{x})]\!]_{(s,h)} = \mathrm{True}$.

Assume $[\![P]\!]((s,h)) \ni r$. Since $\{A\}P\{B\}$ is true, $r \neq$ abort and $[\![B]\!]_r = \mathrm{True}$. Hence we have $[\![P]\!]((s,h)) \ni r$ implies $r \neq$ abort and $[\![B]\!]_r = \mathrm{True}$. By Lemma 5.2.12 (2), we have $[\![\mathrm{W}_{P,B}(\overrightarrow{x})]\!]_{(s,h)} = \mathrm{True}$.

Hence $A \to \mathrm{W}_{P,B}(\overrightarrow{x})$ is true. $\square$

The following lemma (1) shows that the inference rule (RULE 10: SUBSTITUTION RULE I) in [3] is derivable in our system.

Lemma 5.3.5 *(1) If $\Gamma \vdash \{A\}P\{B\}$ is provable, $\overrightarrow{u}$ and $\overrightarrow{w}$ are mutually exclusive, and $\overrightarrow{u}, \overrightarrow{w} \notin EFV(P)$, then $\Gamma \vdash \{A[\overrightarrow{u} := \overrightarrow{w}]\}P\{B[\overrightarrow{u} := \overrightarrow{w}]\}$ is provable.*

(2) If $\Gamma \vdash \{A\}P\{B\}$ is true, $\overrightarrow{u}$ and $\overrightarrow{w}$ are mutually exclusive, and $\overrightarrow{u}, \overrightarrow{w} \notin EFV(P)$, then $\Gamma \vdash \{A[\overrightarrow{u} := \overrightarrow{w}]\}P\{B[\overrightarrow{u} := \overrightarrow{w}]\}$ is true.

Proof. (1) Assume $\Gamma \vdash \{A\}P\{B\}$, $\overrightarrow{u}$ and $\overrightarrow{w}$ are mutually exclusive, and $\overrightarrow{u}, \overrightarrow{w} \notin \mathrm{EFV}(P)$. Then by (Inv-Conj), $\Gamma \vdash \{A \wedge \overrightarrow{u} = \overrightarrow{w}\}P\{B \wedge \overrightarrow{u} = \overrightarrow{w}\}$. We have $B \wedge \overrightarrow{u} = \overrightarrow{w} \to B[\overrightarrow{u} := \overrightarrow{w}]$. Then by (Conseq), $\Gamma \vdash \{A \wedge \overrightarrow{u} = \overrightarrow{w}\}P\{B[\overrightarrow{u} := \overrightarrow{w}]\}$. Then by (Exists), $\Gamma \vdash \{\exists \overrightarrow{u}(A \wedge \overrightarrow{u} = \overrightarrow{w})\}P\{B[\overrightarrow{u} := \overrightarrow{w}]\}$. We have $A[\overrightarrow{u} := \overrightarrow{w}] \to \exists \overrightarrow{u}(A \wedge \overrightarrow{u} = \overrightarrow{w})$. Then by (Conseq), $\Gamma \vdash \{A[\overrightarrow{u} := \overrightarrow{w}]\}P\{B[\overrightarrow{u} := \overrightarrow{w}]\}$.

(2) Assume $\Gamma \vdash \{A\}P\{B\}$ is true, $\overrightarrow{u}$ and $\overrightarrow{w}$ are mutually exclusive, and $\overrightarrow{u}, \overrightarrow{w} \notin \mathrm{EFV}(P)$. Let A_1 be $A[\overrightarrow{u} := \overrightarrow{w}]$ and B_1 be $B[\overrightarrow{u} := \overrightarrow{w}]$.

Assume $[\![A_1]\!]_{(s_1,h_1)} = \mathrm{True}$ and $r_2 \in [\![P]\!]((s_1, h_1))$. We will show $r_2 \neq \mathrm{abort}$ and $[\![B_1]\!]_{r_2} = \mathrm{True}$.

Let s_1' be $s_1[\overrightarrow{u} := s_1(\overrightarrow{w})]$. Then $[\![A]\!]_{(s_1',h_1)} = \mathrm{True}$. Since $s_1 =_{\mathrm{EFV}(P)} s_1'$, by Lemma 4.2.16 (1), we have $r_2 \neq \mathrm{abort}$. Let r_2 be (s_2, h_2) and s_2' be $[s_2, s_1', \mathrm{EFV}(P)]$. By Lemma 4.2.16 (2), we have $(s_2', h_2) \in [\![P]\!]((s_1', h_1))$. Then $[\![B]\!]_{(s_2',h_2)} = \mathrm{True}$.

We will show $s_2' = s_2[\overrightarrow{u} := s_2(\overrightarrow{w})]$. Since $s_1' = s_1[\overrightarrow{u} := s_1(\overrightarrow{w})]$ by the definition and $s_1[\overrightarrow{u} := s_1(\overrightarrow{w})] =_{\mathrm{EFV}(P)^c} s_2[\overrightarrow{u} := s_2(\overrightarrow{w})]$ by $\overrightarrow{w} \notin \mathrm{EFV}(P)$, we have $s_1' =_{\mathrm{EFV}(P)^c} s_2[\overrightarrow{u} := s_2(\overrightarrow{w})]$. By combining it with $s_2[\overrightarrow{u} := s_2(\overrightarrow{w})] = [s_2, s_2[\overrightarrow{u} := s_2(\overrightarrow{w})], \mathrm{EFV}(P)]$ by $\overrightarrow{u} \notin \mathrm{EFV}(P)$, we have $s_2[\overrightarrow{u} := s_2(\overrightarrow{w})] = [s_2, s_1', \mathrm{EFV}(P)]$. Hence $s_2[\overrightarrow{u} := s_2(\overrightarrow{w})] = s_2'$ by the definition of s_2'.

Hence $[\![B]\!]_{(s_2[\overrightarrow{u}:=s_2(\overrightarrow{w})],h_2)} = \mathrm{True}$. Therefore $[\![B_1]\!]_{(s_2,h_2)} = \mathrm{True}$. $\square$

Let V be $\bigcup_{i=1}^{n_{proc}} \mathrm{EFV}(R_i)$. We next take the sequence of mutually distinct variables $\overrightarrow{y} = y_1, \ldots, y_l$ such that $\{y_1, \ldots, y_l\} = V$. Then we choose the sequence of variables $\overrightarrow{z} = z_1, \ldots, z_l$, $\overrightarrow{z}' = z_1', \ldots, z_l'$, and variable x_h, x_h' such that $\overrightarrow{y}$ and they are all mutually distinct. From now, we assume that for given assertions and programs that we will discuss, we fix some sequence $\overrightarrow{x}$ of variables that contains V, the free variables of the assertions, the extended free variables of the programs, and $\overrightarrow{z}', x_h'$. Finally we use this $\overrightarrow{x}$ for construction of the weakest preconditions and the variables $\overrightarrow{y}, \overrightarrow{z}, x_h$ for construction of the strongest postconditions.

We will explain the roles of these variables. For simplicity we sometimes write $\overrightarrow{x}$ for the set of elements contained in the sequence $\overrightarrow{x}$. The expressiveness theorem

(Theorem 5.2.13) assumed the coding of a store by using some fixed interesting variables $\overrightarrow{x}$. The variables in V in $\overrightarrow{x}$ are necessary to define the weakest preconditions of procedures since procedures are defined with V. The variables $\overrightarrow{z}'$, x'_h are necessary to define the weakest preconditions of some assertions that are used in Lemma 5.3.8. After fixing the set $\overrightarrow{x}$ of variables, we only consider assertions and programs whose free variables and extended free variables are contained in the set $\overrightarrow{x} - V \cup \overrightarrow{z}' \cup \{x'_h\}$. Since our choice of the set $\overrightarrow{x}$ of variables is arbitrary, this works for arbitrary given assertions and programs. By these definitions, we also have the variables $\overrightarrow{z}$, x_h that are not in $\overrightarrow{x}$ and are used in Definition 7.6.

The next definition plays a key role in our completeness proof.

Definition 5.3.6 *We define W_i as $W_{R_i,True}(\overrightarrow{y})$, G_i as $\overrightarrow{y} = \overrightarrow{z} \wedge Heap(x_h) \wedge W_i$, and S_i as $S_{G_i,R_i}(\overrightarrow{y}, \overrightarrow{z}, x_h)$. We also define F_i as $\{G_i\}R_i\{S_i\}$.*

Here G_i has three purposes. First, the expression $\overrightarrow{y} = \overrightarrow{z}$ enables us to describe complete information of a given store, which is inspired from [3]. Second, the expression $\mathrm{Heap}(x_h)$ enables us to describe complete information of a given heap. Third, W_i ensures abort-free execution of the program, which enables us to use the strongest postcondition.

5.3.3 Main Proofs

The following Lemma is the key lemma to prove the completeness theorem.

Lemma 5.3.7 *If $\{A\}P\{B\}$ is true then $F_1, \ldots, F_{n_{proc}} \vdash \{A\}P\{B\}$ is provable.*

Proof. We will prove it by induction on P. We will consider the cases of P.

Case 1. P is $x := e$.

We will show that $A \to B[x := e]$ is true. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. Let n be $[\![e]\!]_s$. We have $[\![x := e]\!]((s,h)) = \{(s_1, h)\}$ where $s_1 = s[x := n]$. Since $\{A\}x := e\{B\}$ is true, $[\![B]\!]_{(s_1,h)} = \mathrm{True}$. Since $[\![B]\!]_{(s_1,h)} = [\![B[x := e]]\!]_{(s,h)}$, we have $[\![B[x := e]]\!]_{(s,h)} = \mathrm{True}$. Hence $A \to B[x := e]$ is true.

By applying the rule (Conseq) to the fact that $A \to B[x := e]$ is true and $\vdash \{B[x := e]\}x := e\{B\}$ from the axiom (Assignment), we have $\vdash \{A\}x := e\{B\}$. Therefore, $F_1, \ldots, F_{n_{proc}} \vdash \{A\}x := e\{B\}$.

Case 2. P is if (b) then (P_1) else (P_2).

Assume that $\{A\}P\{B\}$ is true. First, we will show that $\{A \wedge b\}P_1\{B\}$ is true. Assume $[\![A \wedge b]\!]_{(s,h)} = \text{True}$ and $[\![P_1]\!]((s, h)) \ni r$. We have $[\![b]\!]_s = [\![b]\!]_{(s,h)} = \text{True}$ by Lemma 5.1.3. Hence $[\![P]\!]((s, h)) = [\![P_1]\!]((s, h)) \ni r$. Then $r \neq$ abort and $[\![B]\!]_r = \text{True}$. Hence $\{A \wedge b\}P_1\{B\}$ is true.

Similarly $\{A \wedge \neg b\}P_2\{B\}$ is true.

By induction hypothesis for P_1 and P_2, we have $F_1, \ldots, F_{n_{proc}} \vdash \{A \wedge b\}P_1\{B\}$ and $F_1, \ldots, F_{n_{proc}} \vdash \{A \wedge \neg b\}P_2\{B\}$. By the rule (If), therefore, we have $F_1, \ldots, F_{n_{proc}} \vdash \{A\}$if (b) then (P_1) else $(P_2)\{B\}$.

Case 3. P is while (b) do (P_1).

Assume that $\{A\}P\{B\}$ is true. Let C be $\mathrm{W}_{P,B}(\overrightarrow{x})$.

We will show that $\{C \wedge b\}P_1\{C\}$ is true. Assume $[\![C \wedge b]\!]_{(s,h)} = \text{True}$ and $[\![P_1]\!]((s, h)) \ni r$. We will show $r \neq$ abort and $[\![C]\!]_r = \text{True}$. We have $[\![b]\!]_s = [\![b]\!]_{(s,h)} = \text{True}$ by Lemma 5.1.3. By the definition of $[\![P]\!]$, we have $[\![P]\!]((s, h)) \supseteq [\![P]\!](r)$. Since $\{C\}P\{B\}$ is true by Lemma 5.2.12 (1), from $[\![C]\!]_{(s,h)} = \text{True}$, we have $[\![P]\!]((s, h)) \not\ni$ abort. Hence $r \neq$ abort. Assume $[\![P]\!](r) \ni r'$. Then $r' \in [\![P]\!]((s, h))$ and we have $r' \neq$ abort and $[\![B]\!]_{r'} = \text{True}$. By Lemma 5.2.12 (2), we have $[\![C]\!]_r = \text{True}$. Hence $\{C \wedge b\}P_1\{C\}$ is true.

By induction hypothesis for P_1, we have $F_1, \ldots, F_{n_{proc}} \vdash \{C \wedge b\}P_1\{C\}$.

By Lemma 5.3.4 with $\{A\}P\{B\}$ being true, $A \to C$ is true.

We will show that $C \wedge \neg b \to B$ is true. Assume $[\![C \wedge \neg b]\!]_{(s,h)} = \text{True}$. We will show $[\![B]\!]_{(s,h)} = \text{True}$. We have $[\![\neg b]\!]_s = [\![\neg b]\!]_{(s,h)} = \text{True}$. Hence $[\![P]\!]((s, h)) = \{(s, h)\}$. Since $\{C\}P\{B\}$ is true by Lemma 5.2.12 (1), from $[\![C]\!]_{(s,h)} = \text{True}$ and $[\![P]\!]((s, h)) = \{(s, h)\}$, we have $[\![B]\!]_{(s,h)} = \text{True}$. Hence $C \wedge \neg b \to B$ is true.

Since $F_1, \ldots, F_{n_{proc}} \vdash \{C \wedge b\}P_1\{C\}$, by the rule (While), we have $F_1, \ldots, F_{n_{proc}} \vdash \{C\}P\{C \wedge \neg b\}$. Since $A \to C$ and $C \wedge \neg b \to B$ are true, by the rule (Conseq), we have $F_1, \ldots, F_{n_{proc}} \vdash \{A\}P\{B\}$.

Case 4. P is $P_1; P_2$.

Assume that $\{A\}P_1; P_2\{B\}$ is true. Let P be $P_1; P_2$ and we can take C to be $\mathrm{W}_{P_2,B}(\overrightarrow{x})$ by theorem 5.2.13. By Lemma 5.2.12 (1), $\{C\}P_2\{B\}$ is true.

We will show that $\{A\}P_1\{C\}$ is true. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$ and $[\![P_1]\!]((s,h)) \ni r$. We will show $r \neq \mathrm{abort}$ and $[\![C]\!]_r = \mathrm{True}$. Since $\{A\}P\{B\}$ is true, $[\![P]\!]((s,h)) \not\ni \mathrm{abort}$. Since $[\![P]\!]((s,h)) \supseteq [\![P_2]\!](r)$ by the definition of $[\![P]\!]$, $r \neq \mathrm{abort}$. We will show $[\![C]\!]_r = \mathrm{True}$. Assume $[\![P_2]\!](r) \ni r_1$. Then $[\![P]\!]((s,h)) \ni r_1$. Then we have $r_1 \neq \mathrm{abort}$ and $[\![B]\!]_{r_1} = \mathrm{True}$. By Lemma 5.2.12 (2), $[\![C]\!]_r = \mathrm{True}$. Hence $\{A\}P_1\{C\}$ is true.

By induction hypothesis for P_1 and P_2, we have $F_1, \ldots, F_{n_{proc}} \vdash \{A\}P_1\{C\}$ and $F_1, \ldots, F_{n_{proc}} \vdash \{C\}P_2\{B\}$. By the rule (Composition), we have therefore $F_1, \ldots, F_{n_{proc}} \vdash \{A\}P_1; P_2\{B\}$.

Case 5. P is $x := \mathrm{cons}(e_1, e_2)$.

Assume $\{A\}x := \mathrm{cons}(e_1, e_2)\{B\}$ is true. Let x' be such that $x' \notin \mathrm{FV}(e_1, e_2, B)$ and C be $\forall x'((x' \mapsto e_1, e_2) \mathrel{-\!\!*} B[x := x'])$.

We will show that $A \to C$ is true. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. Fix n. Let $s' = s[x' := n]$. We will show $[\![(x' \mapsto e_1, e_2) \mathrel{-\!\!*} B[x := x']]\!]_{(s',h)} = \mathrm{True}$. Assume $[\![x' \mapsto e_1, e_2]\!]_{(s',h_1)} = \mathrm{True}$. Then $h_1 = \emptyset[n := [\![e_1]\!]_{s'}, n+1 := [\![e_2]\!]_{s'}]$. Assume that $h + h_1$ exists. Let $h_2 = h + h_1$. Now we will prove that $[\![B[x := x']]\!]_{(s',h_2)} = \mathrm{True}$.

Let $s_1 = s[x := n]$. Since $[\![x := \mathrm{cons}(e_1, e_2)]\!]((s,h)) \ni (s_1, h_2)$ by definition, we have $[\![B]\!]_{(s_1,h_2)} = \mathrm{True}$. Since $[\![B[x := x']]\!]_{(s',h_2)} = [\![B]\!]_{(s_1,h_2)}$ by $x' \notin \mathrm{FV}(B)$, we have $[\![B[x := x']]\!]_{(s',h_2)} = \mathrm{True}$. Therefore $[\![(x' \mapsto e_1, e_2) \mathrel{-\!\!*} B[x := x']]\!]_{(s',h)} = \mathrm{True}$.

Hence $[\![(x' \mapsto e_1, e_2) \mathrel{-\!\!*} B[x := x']]\!]_{(s',h)} = \mathrm{True}$ for all n. Hence $[\![\forall x'((x' \mapsto e_1, e_2) \mathrel{-\!\!*} B[x := x'])]\!]_{(s,h)} = \mathrm{True}$. Hence $A \to C$ is true.

Since $\vdash \{C\}x := \mathrm{cons}(e_1, e_2)\{B\}$ is provable by the axiom (cons) and $A \to C$ is true, we have $\vdash \{A\}x := \mathrm{cons}(e_1, e_2)\{B\}$ by the rule (Conseq). Therefore, by

(WEAKENING) $F_1, \ldots, F_{n_{proc}} \vdash \{A\}x := \mathrm{cons}(e_1, e_2)\{B\}$.

Case 6. P is $x := [e]$.

Assume that $\{A\}x := [e]\{B\}$ is true. Let $x' \notin \mathrm{FV}(e, B)$, and C be $\exists x'(e \mapsto x' * (e \mapsto x' -\!\!* B[x := x']))$.

We will show $A \to C$. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. We will show $[\![C]\!]_{(s,h)} = \mathrm{True}$.

Let n be $[\![e]\!]_s$. Since $\{A\}P\{B\}$ is true, $[\![P]\!]((s,h)) \not\ni \mathrm{abort}$. Hence $n \in \mathrm{Dom}(h)$. Let $h(n) = n_1$. We have $[\![P]\!]((s,h)) = \{(s_1, h)\}$ and $[\![B]\!]_{(s_1,h)} = \mathrm{True}$ where $s_1 = s[x := n_1]$. Let $h_1 = h|_{\{n\}}$, $h_2 = h|_{\mathrm{Dom}(h)-\{n\}}$, and $s' = s[x' := n_1]$. Then $h = h_1 + h_2$.

We have $[\![e \mapsto x']\!]_{(s',h_1)} = \mathrm{True}$ since $[\![e]\!]_{s'} = [\![e]\!]_s = n$ by $x' \notin \mathrm{FV}(e)$.

We will show $[\![e \mapsto x' -\!\!* B[x := x']]\!]_{(s',h_2)} = \mathrm{True}$. Assume $[\![e \mapsto x']\!]_{(s',h_1')} = \mathrm{True}$ and $h_2 + h_1'$ exists. We have $h_1 = h_1'$. Hence $h_1' + h_2 = h$. From $[\![B]\!]_{(s_1,h)} = [\![B[x := x']]\!]_{(s',h)}$ by $x' \notin \mathrm{FV}(B)$ and $[\![B]\!]_{(s_1,h)} = \mathrm{True}$, we have $[\![B[x := x']]\!]_{(s',h_2+h_1')} = \mathrm{True}$. Hence $[\![e \mapsto x' -\!\!* B[x := x']]\!]_{(s',h_2)} = \mathrm{True}$.

Combining them, we have $[\![e \mapsto x' * (e \mapsto x' -\!\!* B[x := x'])]\!]_{(s',h)} = \mathrm{True}$. Hence $[\![C]\!]_{(s,h)} = \mathrm{True}$. Hence $A \to C$ is true.

By the axiom (LOOKUP), $\vdash \{C\}P\{B\}$ is provable. Since $A \to C$ is true, by the rule (CONSEQ), we have $\vdash \{A\}P\{B\}$. Therefore, $F_1, \ldots, F_{n_{proc}} \vdash \{A\}x := [e]\{B\}$.

Case 7. P is $[e_1] := e_2$.

Assume that $\{A\}[e_1] := e_2\{B\}$ is true. Let $x \notin \mathrm{FV}(e_1)$ and C be $(\exists x(e_1 \mapsto x)) * (e_1 \mapsto e_2 -\!\!* B)$.

We will show that $A \to C$ is true. Assume $[\![A]\!]_{(s,h)} = \mathrm{True}$. We will show $[\![C]\!]_{(s,h)} = \mathrm{True}$.

Let $n_1 = [\![e_1]\!]_s$. Since $\{A\}P\{B\}$ is true, $[\![P]\!]((s,h)) \not\ni \mathrm{abort}$. Hence $n_1 \in \mathrm{Dom}(h)$. Let $h_2 = h|_{\{n_1\}}$ and $h_3 = h|_{\mathrm{Dom}(h)-\{n_1\}}$. Now we will show $[\![\exists x(e_1 \mapsto x)]\!]_{(s,h_2)} = \mathrm{True}$ and $[\![e_1 \mapsto e_2 -\!\!* B]\!]_{(s,h_3)} = \mathrm{True}$.

Let n_2 be $h_2(n_1)$. Then $[\![e_1 \mapsto x]\!]_{(s[x:=n_2],h_2)} = \mathrm{True}$. Then $[\![\exists x(e_1 \mapsto x)]\!]_{(s,h_2)} =$

True.

Assume $[\![e_1 \mapsto e_2]\!]_{(s,h_4)} = \text{True}$. Then $h_4 = \emptyset[[\![e_1]\!]_s := [\![e_2]\!]_s]$. By definition $[\![P]\!]((s,h)) = \{(s,h_1)\}$ where $h_1 = h_4 + h_3$. Then $[\![B]\!]_{(s,h_1)} = \text{True}$. Then $[\![e_1 \mapsto e_2 \mathrel{-\!\!*} B]\!]_{(s,h_3)} = \text{True}$.

Hence $A \to C$ is true.

By the axiom (Mutation), $\vdash \{C\}P\{B\}$ is provable. Since $A \to C$ is true, by the rule (Conseq), we have $\vdash \{A\}P\{B\}$. Therefore, $F_1, \dots, F_{n_{proc}} \vdash \{A\}[e_1] := e_2\{B\}$.

Case 8. P is $\text{dispose}(e)$.

Assume that $\{A\}\text{dispose}(e)\{B\}$ is true. Let $x \notin \text{FV}(e)$ and C be $(\exists x(e \mapsto x)) * B$.

We will show that $A \to C$ is true. Assume $[\![A]\!]_{(s,h)} = \text{True}$. We will show $[\![C]\!]_{(s,h)} = \text{True}$. Let $n = [\![e]\!]_s$. Since $\{A\}P\{B\}$ is true, $[\![P]\!]((s,h)) \not\ni \text{abort}$. Hence $n \in \text{Dom}(h)$. Hence $[\![P]\!]((s,h)) = \{(s,h_1)\}$ and $[\![B]\!]_{(s,h_1)} = \text{True}$ where $h_1 = h|_{\text{Dom}(h)-\{n\}}$. Let $n_1 = h(n)$, $h_2 = \emptyset[n := n_1]$, and $s' = s[x := n_1]$. We have $h = h_1 + h_2$. Since $[\![e]\!]_{s'} = [\![e]\!]_s = n$ by $x \notin \text{FV}(e)$, we have $[\![e \mapsto x]\!]_{(s',h_2)} = \text{True}$. Hence $[\![\exists x(e \mapsto x)]\!]_{(s,h_2)} = \text{True}$. Hence $[\![C]\!]_{(s,h)} = \text{True}$. Hence $A \to C$ is true.

By the axiom (Dispose), $\vdash \{C\}P\{B\}$ is provable. Since $A \to C$ is true, by the rule (Conseq), we have $\vdash \{A\}P\{B\}$. Therefore, $F_1, \dots, F_{n_{proc}} \vdash \{A\}\text{dispose}(e)\{B\}$.

Case 9. P is R_i.

Assume that $\{A\}R_i\{B\}$ is true. We have $F_1, \dots, F_{n_{proc}} \vdash F_i$. Note that the variables in $\overrightarrow{z}$,x_h,$\text{FV}(A) \cup \text{FV}(B) \cup \text{EFV}(R_i)$ are mutually distinct according to our global assumption. By the rule (Inv-Conj),

$$F_1, \dots, F_{n_{proc}} \vdash \{G_i \wedge \text{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}R_i\{S_i \wedge \text{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}$$

since $\text{FV}(\text{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)) \cap \text{Mod}(R_i) = \emptyset$.

We will prove $S_i \wedge \text{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h) \to B$. Assume that $[\![S_i \wedge \text{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s',h')}$ is true. Then $[\![S_i]\!]_{(s',h')}$ and $[\![\text{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s',h')}$ are true. By Lemma 5.3.2 (2), we have s, h such that $[\![G_i]\!]_{(s,h)}$ is true and

$[\![R_i]\!]((s,h)) \ni (s',h')$.

Now we will show $[\![\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s,h)} = \mathrm{True}$ by contradiction. Assume $[\![\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s,h)} = \mathrm{False}$. Then $[\![\neg\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s,h)} = \mathrm{True}$. By (Inv-Conj),

$$F_1, \ldots, F_{n_{proc}} \vdash \{G_i \wedge \neg\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}R_i\{S_i \wedge \neg\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}.$$

By Theorem 5.1.8, $F_1, \ldots, F_{n_{proc}} \vdash \{G_i \wedge \neg\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}R_i\{S_i \wedge \neg\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}$ is true. Since $F_1, \ldots, F_{n_{proc}}$ are true by Lemma 5.3.2 (1) with the fact that $\{G_i\}R_i\{\mathrm{True}\}$ is true by Lemma 5.2.12 (1), $\{G_i \wedge \neg\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}R_i\{S_i \wedge \neg\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}$ is true.

By the definition of the truth of asserted programs, we have $[\![\neg\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s',h')} = \mathrm{True}$. Then $[\![\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s',h')} = \mathrm{False}$. But it contradicts with $[\![\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s',h')} = \mathrm{True}$. Thus $[\![\mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)]\!]_{(s,h)} = \mathrm{True}$.

Then $s(\overrightarrow{z}) = s(\overrightarrow{y})$ and $[\![\mathrm{Heap}(x_h) \wedge W_i \wedge \mathrm{HEval}_A(x_h)]\!]_{(s,h)}$ is true. Since $\mathrm{Heapcode}(s(x_h), h)$ holds and $[\![\mathrm{HEval}_A(x_h)]\!]_{(s,h)}$ is true, by Lemma 5.2.7 (3), we have $[\![A]\!]_{(s,h)} = \mathrm{True}$.

Since $\{A\}R_i\{B\}$ is true, $[\![B]\!]_{(s',h')}$ is true. Then $S_i \wedge \mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h) \to B$ is true. Then by (Conseq) rule,

$$F_1, \ldots, F_{n_{proc}} \vdash \{G_i \wedge \mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h)\}R_i\{B\}$$

is provable. By (Exists) rule,

$$F_1, \ldots, F_{n_{proc}} \vdash \{\exists \overrightarrow{z} x_h(\overrightarrow{y} = \overrightarrow{z} \wedge \mathrm{Heap}(x_h) \wedge W_i \wedge \mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h))\}R_i\{B\}.$$

Naturally $\{A\}R_i\{\mathrm{True}\}$ is true. Then by Lemma 5.3.4, $A \to W_i$ is true. By Lemma 5.3.3, $A \to \exists \overrightarrow{z} x_h(\overrightarrow{y} = \overrightarrow{z} \wedge \mathrm{Heap}(x_h) \wedge \mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h))$ is true and then we have $A \to \exists \overrightarrow{z} x_h(\overrightarrow{y} = \overrightarrow{z} \wedge \mathrm{Heap}(x_h) \wedge \mathrm{HEval}_{A[\overrightarrow{y}:=\overrightarrow{z}]}(x_h) \wedge W_i)$. Then by (conseq) rule,

$$F_1, \ldots, F_{n_{proc}} \vdash \{A\}R_i\{B\}$$

, which was to be proved. □

Next Lemma shows that the hypothesis $F_1, \ldots, F_{n_{proc}}$ used in lemma 5.3.7 are provable in the our system.

Lemma 5.3.8 *$\vdash F_i$ is provable for $i = 1, \ldots, n_{proc}$.*

Proof. Fix i. Note that we have fresh variables $\overrightarrow{z}'$ and x'_h according to our global assumption. Let G'_i be $G_i[\overrightarrow{z} := \overrightarrow{z}', x_h := x'_h]$ and S'_i be $S_i[\overrightarrow{z} := \overrightarrow{z}', x_h := x'_h]$. By Lemma 5.2.12 (1), $\{W_i\}R_i\{\mathrm{True}\}$ is true. Then $\{G_i\}R_i\{\mathrm{True}\}$ is true. Then by Lemma 5.3.2 (1), $\{G_i\}R_i\{S_i\}$ is true. By Lemma 4.2.12, $[\![R_i]\!] = [\![Q_i]\!]$ where Q_i is the body of R_i. Hence $\{G_i\}Q_i\{S_i\}$ is true. By Lemma 5.3.5 (2), $\{G'_i\}Q_i\{S'_i\}$ is true. Then by Lemma 5.3.7, $F_1, \ldots, F_{n_{proc}} \vdash \{G'_i\}Q_i\{S'_i\}$ is provable. By Lemma 5.3.5 (1), $F_1, \ldots, F_{n_{proc}} \vdash \{G_i\}Q_i\{S_i\}$ is provable. By (Recursion) rule, $\vdash F_i$ is provable. □

The following theorem is our central result of this paper. It says that our system is complete.

Theorem 5.3.9 *If $\{A\}P\{B\}$ is true then $\vdash \{A\}P\{B\}$ is provable.*

Proof. Assume $\{A\}P\{B\}$ is true. Then by Lemma 5.3.7, $F_1, \ldots, F_{n_{proc}} \vdash \{A\}P\{B\}$ is provable. By Lemma 5.3.8, $\vdash F_i$ is provable for $i = 1, \ldots, n_{proc}$. By (Cut), $\vdash \{A\}P\{B\}$ is also provable. □

6

Admissibility of Frame Rules

We have the soundness for $\Gamma \vdash \{A\}P\{B\}$ (Theorem 5.1.8), but we have the completeness only for $\vdash \{A\}P\{B\}$ (Theorem 5.3.9). Hence for sequents of the shape $\vdash \{A\}P\{B\}$, a rule is sound if and only if the rule is admissible. On the other hand, for rules that use sequents of the shape $\Gamma \vdash \{A\}P\{B\}$, this equivalence may fail. This section shows

- the ordinary frame rule is sound, but not admissible,
- the uniform hypothetical frame rule is not sound nor admissible,
- the hypothetical frame rule is not sound nor admissible,
- the hypothesis-free frame rule is sound and admissible,
- the conjunction rule is sound, but not admissible.

The ordinary frame rule and the hypothetical frame rule are important for the local reasoning in separation logic as discussed in [7, 17, 23]. It is because we can use the hypothetical judgments as for the specifications of the procedures in terms of their actually used memory to reason a program in an extended memory space. A natural question is how important these rules are for the completeness of our system. Since we can achieve the completeness for asserted programs without hypothesis in our system without these rules, indeed they are not necessary for the completeness. However when we think completeness for asserted programs with a hypothesis, they may be important. In fact, the ordinary frame rule is not admissible in our system, and we will show it in this section. Moreover, the uniform hypothetical frame rule, where all the specifications in the hypothesis are extended with the invariant uniformly, is neither sound nor admissible. As a consequence, the hypothetical frame rule is not admissible as well since the frame rules mentioned above are special forms of it. However, a frame rule with an empty hypothesis, called the hypothesis-free frame rule, is admissible and sound in our system. Another interesting question is about the role of the conjunction rule [17] in our system. It is clear that the conjunction rule is not necessary for the completeness for asserted programs without a hypothesis. In fact, the conjunction rule is not admissible in the system. In this section, we will investigate the soundness and the admissibility of these rules.

6.1 Frame Rules

6.1.1 Ordinary Frame Rule

Definition 6.1.1 *The* Ordinary Frame Rule *is defined as -*

$$\frac{\Gamma \vdash \{A\}P\{B\}}{\Gamma \vdash \{A * C\}P\{B * C\}} \ (FV(C) \cap Mod(P) = \emptyset)$$

The following lemmas are used to prove the soundness of the Ordinary Frame Rule.

Lemma 6.1.2 *Suppose $P \in \mathcal{L}^-$. If $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]^-(s, h_1) \not\ni$ abort then $h' = h'_1 + h_2$ and $[\![P]\!]^-((s, h_1)) \ni (s', h'_1)$ for some h'_1.*

Proof. Proved by induction on P. We will consider the cases of P.

Case 1. P is $x := e$.

Assume $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$. Then $h' = h_1 + h_2$ by definition. Take h'_1 to be h_1. Then $h' = h'_1 + h_2$ and $[\![P]\!]^-((s, h_1)) \ni (s', h'_1)$.

Case 2. P is if (b) then (P_1) else (p_2).

Assume $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]^-(s, h_1) \not\ni$ abort.

Case $[\![b]\!]_s =$ True. Then $[\![P_1]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P_1]\!]^-(s, h_1) \not\ni$ abort by definition. Then we have h'_1 such that $h' = h'_1 + h_2$ and $[\![P_1]\!]^-((s, h_1)) \ni (s', h'_1)$ by induction hypothesis. Then $[\![P]\!]^-((s, h_1)) \ni (s', h'_1)$ for some h'_1 by definition.

Case $[\![b]\!]_s =$ False can be shown as above.

Case 3. P is while (b) do (P_1).

Assume $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]^-((s, h_1)) \not\ni$ abort. By Proposition 4.2.4, we have $m \geq 0, s''_0, \ldots, s''_m, h''_0, \ldots, h''_m$ such that $(s''_0, h''_0) = (s, h_1 + h_2)$, $(s', h') = (s''_m, h''_m)$, $[\![b]\!]_{s''_i} =$ True, $[\![P_1]\!]^-((s''_i, h''_i)) \ni (s''_{i+1}, h''_{i+1})$ for $0 \leq i < m$, and $[\![b]\!]_{s''_m} =$ False. Take $h'''_0 = h_1$. Then $[\![P_1]\!]^-((s''_i, h'''_i)) \not\ni$ abort and by induction hypothesis we have h'''_{i+1} such that $h''_{i+1} = h'''_{i+1} + h_2$ and $[\![P_1]\!]^-((s''_i, h'''_i)) \ni (s''_{i+1}, h'''_{i+1})$ for $0 \leq i < m$. Take h'_1 be h'''_m. Then by Proposition 4.2.4, $[\![P]\!]^-((s, h_1)) \ni (s', h'_1)$.

Case 4. P is $P_1; P_2$.

Assume $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]^-(s, h_1) \not\ni$ abort. Then $[\![P_1]\!]^-((s, h_1)) \not\ni$ abort and we have s'', h'' such that $[\![P_1]\!]^-((s, h_1 + h_2)) \ni (s'', h'')$ and $[\![P_2]\!]^-((s'', h'')) \ni (s', h')$. By induction hypothesis, we have h''_1 such that $h'' = h''_1 + h_2$ and $[\![P_1]\!]^-((s, h_1)) \ni (s'', h''_1)$. Then $[\![P_2]\!]^-((s, h''_1)) \not\ni$ abort since $[\![P]\!]^-((s, h_1)) \not\ni$ abort. By induction hypothesis, we have h'_1 such that $h' = h'_1 + h_2$ and $[\![P_2]\!]^-((s'', h''_1)) \ni (s', h'_1)$. Then $[\![P]\!]^-((s, h_1)) \ni (s', h'_1)$.

Case 5. P is skip.

Its proof is immediate.

Case 6. P is $x := \mathrm{cons}(e_1, e_2)$.

Assume $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]^-((s, h_1)) \not\ni$ abort. By definition, $h' = (h_1+h_2)[n := [\![e_1]\!]_s, n+1 := [\![e_2]\!]_s]$ where $n > 0, n, n+1 \notin \mathrm{Dom}(h_1+h_2)$. Then $h' = (h_1[n := [\![e_1]\!]_s, n+1 := [\![e_2]\!]_s]) + h_2$. Take h_1' to be $h_1[n := [\![e_1]\!]_s, n+1 := [\![e_2]\!]_s]$. Then $h' = h_1' + h_2$ and $[\![P]\!]^-((s, h_1)) \ni (s', h_1')$ by definition.

Case 7. P is $x := [e]$.

Assume $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]^-((s, h_1)) \not\ni$ abort. Then $h' = h_1 + h_2$ and $[\![e]\!]_s \in \mathrm{Dom}(h_1)$ by definition. Take h_1' to be h_1. Then $h' = h_1' + h_2$ and $[\![P]\!]^-((s, h_1)) \ni (s', h_1')$.

Case 8. P is $[e_1] := e_2$.

Assume $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]^-((s, h_1)) \not\ni$ abort. By definition, $h' = (h_1 + h_2)[[\![e_1]\!]_s := [\![e_2]\!]_s]$ where $[\![e_1]\!]_s \in \mathrm{Dom}(h_1)$. Then $h' = (h_1[[\![e_1]\!]_s := [\![e_2]\!]_s]) + h_2$. Take h_1' to be $h_1[[\![e_1]\!]_s := [\![e_2]\!]_s]$. Then $h' = h_1' + h_2$ and $[\![P]\!]^-((s, h_1)) \ni (s', h_1')$ by definition.

Case 9. P is $\mathrm{dispose}(e)$.

Assume $[\![P]\!]^-((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]^-((s, h_1)) \not\ni$ abort. By definition, $h' = (h_1 + h_2)|_{\mathrm{Dom}(h_1+h_2)-\{[\![e]\!]_s\}})$ where $[\![e]\!]_s \in \mathrm{Dom}(h_1)$. Then $h' = (h_1|_{\mathrm{Dom}(h_1)-\{[\![e]\!]_s\}}) + h_2$. Take h_1' to be $h_1|_{\mathrm{Dom}(h_1)-\{[\![e]\!]_s\}}$. Then $h' = h_1' + h_2$ and $[\![P]\!]^-((s, h_1)) \ni (s', h_1')$ by definition. $\square$

Lemma 6.1.3 *Suppose $P \in \mathcal{L}$. If $[\![P]\!]((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!](s, h_1) \not\ni$ abort then $h' = h_1' + h_2$ and $[\![P]\!]((s, h_1)) \ni (s', h_1')$ for some h_1'.*

Proof. Assume $[\![P]\!]((s, h_1 + h_2)) \ni (s', h')$ and $[\![P]\!]((s, h_1)) \not\ni$ abort. By definition, $[\![P^{(k)}]\!]^-((s, h_1 + h_2)) \ni (s', h')$ for some k and $[\![P^{(k')}]\!]^-((s, h_1)) \not\ni$ abort for all k'. Hence $[\![P^{(k)}]\!]^-((s, h_1)) \not\ni$ abort. By Lemma 6.1.2, $h' = h_1' + h_2$ and $[\![P^{(k)}]\!]^-((s, h_1)) \ni (s', h_1')$ for some h_1'. Hence by definition $[\![P]\!]((s, h_1)) \ni (s', h_1')$. $\square$

Below we show that the Ordinary Frame Rule is sound.

Proposition 6.1.4 *The* Ordinary Frame Rule *is sound. Namely, if* $\Gamma \vdash \{A\}P\{B\}$ *is true then* $\Gamma \vdash \{A * C\}P\{B * C\}$ *is true where* $Mod(P) \cap FV(C) = \emptyset$*.*

Proof. Assume $\Gamma \vdash \{A\}P\{B\}$ is true. Assume Γ is true, $[\![A]\!]_{(s,h_1)} = \text{True}$, $[\![C]\!]_{(s,h_2)} = \text{True}$, and $[\![P]\!]((s, h_1 + h_2)) \ni (s', h')$. We will show that $[\![B * C]\!]_{(s',h')} = \text{True}$.

Then $\{A\}P\{B\}$ is true since $\Gamma \vdash \{A\}P\{B\}$ and Γ are true. Then $[\![P]\!]((s, h_1)) \not\ni \text{abort}$. By Lemma 6.1.3, $h' = h_1' + h_2$ and $[\![P]\!]((s, h_1)) \ni (s', h_1')$ for some h_1'. Then $[\![B]\!]_{(s',h_1')} = \text{True}$. Then $[\![C]\!]_{(s',h_2)} = \text{True}$ since $s =_{\mathrm{FV}(C)} s'$. Then by definition, $[\![B * C]\!]_{(s',h')} = \text{True}$. Therefore, $\Gamma \vdash \{A * C\}P\{B * C\}$ is true. □

Below we will show that the Ordinary Frame Rule is not admissible.

Lemma 6.1.5 *If* $\{A\}P\{B\}$ *is false,* P *is atomic, and* $\Gamma \vdash \{A\}P\{B\}$ *has a proof with* $(\leq n)$ *cut rules, then* $\Gamma \vdash \{A\}P\{B\}$ *is provable only by (*Identity*), (*Weakening*), (*Exists*), (*Inv-Conj*), and (*Conseq*).*

Proof. By induction on n, we will show that, if $\Gamma \vdash \{A\}P\{B\}$ is provable with $(\leq n)$ cut rules then $\Gamma \vdash \{A\}P\{B\}$ has a proof only by (Identity), (Weakening), (Exists), (Inv-Conj), and (Conseq).

Assume that $\Gamma \vdash \{A\}P\{B\}$ has some proof with $(\leq n)$ cut rules. We consider the cases of n.

Case 1. $n = 0$.

The proof does not have (If), (While), (Composition), and (Recursion) since P is atomic. Hence the proof has some first axiom (Skip), (Assignment), (Cons), (Lookup), (Mutation), and (Dispose) and it is followed by some of (Weakening), (Conseq), (Exists), (Inv-Conj), or (Identity). Since $\{A\}P\{B\}$ is False, by Theorem 5.1.8, the first axiom is (Identity).

Case 2. $n > 0$.

If the last rule is (Weakening), (Conseq), (Exists), or (Inv-Conj), we can move it upward. Hence we can assume that the last rule is (Cut). Then we have $\Gamma \vdash \{A'\}P'\{B'\}$ and $\Gamma \cup \{\{A'\}P'\{B'\}\} \vdash \{A\}P\{B\}$ for some $\{A'\}P'\{B'\}$. Then

$\Gamma \cup \{\{A'\}P'\{B'\}\} \vdash \{A\}P\{B\}$ is provable with $(< n)$ cut rules. By induction hypothesis, it is provable only by (IDENTITY), (WEAKENING), (EXISTS), (INV-CONJ), and (CONSEQ).

Case 2.1. The (IDENTITY) does not use $\{A'\}P'\{B'\}$. Then $\{A'\}P'\{B'\}$ is introduced by (WEAKENING). Hence $\Gamma \vdash \{A\}P\{B\}$ is provable by (IDENTITY), (WEAKENING), (EXISTS), (INV-CONJ), and (CONSEQ).

Case 2.2. The (IDENTITY) uses $\{A'\}P'\{B'\}$.

Since $\{A\}P\{B\}$ is false, by Theorem 5.1.8, $\{A'\}P'\{B'\}$ is false. Since $P' = P$ and $\Gamma \vdash \{A'\}P\{B'\}$ is provable with $(< n)$ cut rules, by induction hypothesis, $\Gamma \vdash \{A'\}P\{B'\}$ is provable only by (IDENTITY), (WEAKENING), (EXISTS), (INV-CONJ), and (CONSEQ).

Since $\{A'\}P'\{B'\}$ is in Γ, combining the proof of $\Gamma \vdash \{A'\}P'\{B'\}$ and the proof of $\Gamma \cup \{\{A'\}P'\{B'\}\} \vdash \{A\}P\{B\}$, $\Gamma \vdash \{A\}P\{B\}$ is provable only by (IDENTITY), (WEAKENING), (EXISTS), (INV-CONJ), and (CONSEQ).

We have shown that if $\Gamma \vdash \{A\}P\{B\}$ is provable with $(\leq n)$ cut rules then $\Gamma \vdash \{A\}P\{B\}$ has a proof only by (IDENTITY), (WEAKENING), (EXISTS), (INV-CONJ), and (CONSEQ). □

Proposition 6.1.6 *The ORDINARY FRAME RULE is not admissible.*

Proof. By letting $\Gamma \vdash \{A\}P\{B\}$ be $\{\{\mathrm{emp}\}x := [1]\{\mathrm{emp}\}\} \vdash \{\mathrm{emp}\}x := [1]\{\mathrm{emp}\}$ in Definition 6.1.1 and C be $2 \mapsto 0$ we will show that it gives a counterexample. By (IDENTITY), $\{\{\mathrm{emp}\}x := [1]\{\mathrm{emp}\}\} \vdash \{\mathrm{emp}\}x := [1]\{\mathrm{emp}\}$ is provable. We will show that $\{\{\mathrm{emp}\}x := [1]\{\mathrm{emp}\}\} \vdash \{\mathrm{emp} * 2 \mapsto 0\}x := [1]\{\mathrm{emp} * 2 \mapsto 0\}$ is not provable.

Assume $\{\{\mathrm{emp}\}x := [1]\{\mathrm{emp}\}\} \vdash \{\mathrm{emp} * 2 \mapsto 0\}x := [1]\{\mathrm{emp} * 2 \mapsto 0\}$ is provable. Here $x := [1]$ is atomic and $\{\mathrm{emp}\}x := [1]\{\mathrm{emp}\}$ is false. By Lemma 6.1.5, $\{\{\mathrm{emp}\}x := [1]\{\mathrm{emp}\}\} \vdash \{\mathrm{emp}*2 \mapsto 0\}x := [1]\{\mathrm{emp}*2 \mapsto 0\}$ is provable by first (IDENTITY) and some of (WEAKENING), (EXISTS), (INV-CONJ), and (CONSEQ). Because of the shape of $\mathrm{emp} * 2 \mapsto 0$, (EXISTS) and (INV-CONJ) are not used. Hence for (CONSEQ), $(\mathrm{emp} * 2 \mapsto 0) \to \mathrm{emp}$ and $\mathrm{emp} \to (\mathrm{emp} * 2 \mapsto 0)$ are used in the proof. But it contradicts since $(\mathrm{emp} * 2 \mapsto 0) \to \mathrm{emp}$ and $\mathrm{emp} \to (\mathrm{emp} * 2 \mapsto 0)$ are false.

Thus, the Ordinary Frame Rule is not admissible. $\square$

6.1.2 Uniform Hypothetical Frame Rule

We define $\Gamma * C$ as $\{\, \{A * C\}P\{B * C\} \mid \{A\}P\{B\} \in \Gamma \,\}$.

We define $\mathrm{Mod}(\Gamma)$ as $\bigcup_{\{A_i\}P_i\{B_i\}\in\Gamma} \mathrm{Mod}(P_i)$.

Definition 6.1.7 *The Uniform Hypothetical Frame Rule is defined as -*

$$\frac{\Gamma \vdash \{A\}P\{B\}}{\Gamma * C \vdash \{A * C\}P\{B * C\}} \; (FV(C) \cap (Mod(P) \cup Mod(\Gamma)) = \emptyset)$$

We will show that the Uniform Hypothetical Frame Ruleis not sound in the following lemma.

Proposition 6.1.8 *The Uniform Hypothetical Frame Rule is not sound.*

Proof. This proof is inspired from the example given in Section 6 of [17]. We will show the claim by a counterexample.

By letting $\Gamma \vdash \{A\}P\{B\}$ be $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{False}\}$ and C be $\neg\mathrm{emp}$ we will show that it gives a counterexample.

$\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{False}\}$ is true. By the Uniform Hypothetical Frame Rule, $\{\{\mathrm{emp} \vee \neg\mathrm{emp} * \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp} * \neg\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp} * \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{False} * \neg\mathrm{emp}\}$ is true. But clearly $\{\{\mathrm{emp} \vee \neg\mathrm{emp} * \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp} * \neg\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp} * \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{False} * \neg\mathrm{emp}\}$ is false. $\square$

Now we will show that the Uniform Hypothetical Frame Rule is not admissible.

Proposition 6.1.9 *The Uniform Hypothetical Frame Rule is not admissible.*

Proof. By letting $\Gamma \vdash \{A\}P\{B\}$ be $\{\{x = 1 \wedge y = x\}\text{skip}\{y = 2\}\} \vdash \{\exists x(x = 1 \wedge y = x)\}\text{skip}\{y = 2\}$ and C be $x = 3$ we will show that it gives a counterexample.

By (Identity), $\{\{x = 1 \wedge y = x\}\text{skip}\{y = 2\}\} \vdash \{x = 1 \wedge y = x\}\text{skip}\{y = 2\}$ is provable. By (Exists), $\{\{x = 1 \wedge y = x\}\text{skip}\{y = 2\}\} \vdash \{\exists x(x = 1 \wedge y = x)\}\text{skip}\{y = 2\}$ is provable. If the uniform hypothetical frame rule were admissible then $\{\{x = 1 \wedge y = x * x = 3\}\text{skip}\{y = 2 * x = 3\}\} \vdash \{\exists x(x = 1 \wedge y = x) * x = 3\}\text{skip}\{y = 2 * x = 3\}$ would be provable. By Theorem 5.1.8, it would be true. Since $x = 1 \wedge y = x * x = 3$ is false, we have $\{x = 1 \wedge y = x * x = 3\}\text{skip}\{y = 2 * x = 3\}$ is true. But $\{\exists x(x = 1 \wedge y = x) * x = 3\}\text{skip}\{y = 2 * x = 3\}$ is false. Therefore, $\{\{x = 1 \wedge y = x * x = 3\}\text{skip}\{y = 2 * x = 3\}\} \vdash \{\exists x(x = 1 \wedge y = x) * x = 3\}\text{skip}\{y = 2 * x = 3\}$ is false. So it would contradict. $\square$

6.1.3 Hypothetical Frame Rule

Definition 6.1.10 *The Hypothetical Frame Rule is defined as -*

$$\frac{\Gamma \cup \Gamma' \vdash \{A\}P\{B\}}{\Gamma \cup (\Gamma' * C) \vdash \{A * C\}P\{B * C\}} \; (FV(C) \cap (Mod(P) \cup Mod(\Gamma')) = \emptyset)$$

The next two propositions are proved by Proposition 6.1.8 and Proposition 6.1.9 respectively, since the uniform hypothetical frame rule is a special case of Hypothetical Frame Rule. Proposition 6.1.12 is proved also by Proposition 6.1.6, since the Ordinary Frame Rule is a special case of hypothetical frame rule.

Proposition 6.1.11 *The Hypothetical Frame Rule is not sound.*

Proposition 6.1.12 *The Hypothetical Frame Rule is not admissible.*

6.1.4 Hypothesis-Free Frame Rule

Definition 6.1.13 *The Hypothesis-free Frame Rule is defined as -*

$$\frac{\vdash \{A\}P\{B\}}{\vdash \{A * C\}P\{B * C\}} \ (FV(C) \cap Mod(P) = \emptyset)$$

Note that the Hypothesis-free Frame Rule is sound since the Ordinary Frame Rule is sound. The following proposition shows that the Hypothesis-free Frame Rule is admissible.

Proposition 6.1.14 *The Hypothesis-free Frame Rule is admissible. Namely, if* $\vdash \{A\}P\{B\}$ *is provable then* $\vdash \{A * C\}P\{B * C\}$ *is provable where* $Mod(P) \cap FV(C) = \emptyset$*.*

Proof. Assume $\vdash \{A\}P\{B\}$ is provable. Then by Theorem 5.1.8, $\vdash \{A\}P\{B\}$ is true. Then by Proposition 6.1.4, $\vdash \{A * C\}P\{B * C\}$ is true. By Theorem 5.3.9, $\vdash \{A * C\}P\{B * C\}$ is provable. □

6.2 Conjunction Rule

Definition 6.2.1 *The Conjunction Rule is defined as -*

$$\frac{\Gamma \vdash \{A\}P\{B\} \quad \Gamma \vdash \{C\}P\{D\}}{\Gamma \vdash \{A \wedge C\}P\{B \wedge D\}}$$

The Conjunction Rule is trivially sound by the definition of semantics of asserted programs. Now we will show that the Conjunction Ruleis not admissible.

Proposition 6.2.2 *The Conjunction Rule is not admissible.*

Proof. This proof is inspired from the example given in Section 6 of [17]. By letting $\Gamma \vdash \{A\}P\{B\}$ be $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}$ and $\Gamma \vdash \{C\}P\{D\}$ be $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp}\}\mathrm{skip}\{\neg\mathrm{emp}\}$ we will show that it gives a counterexample.

We have $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}$ by (Identity). Then $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}$ by (Conseq) since $\neg\mathrm{emp} \to \mathrm{emp} \vee \neg\mathrm{emp}$. Again, $\vdash \{\neg\mathrm{emp}\}\mathrm{skip}\{\neg\mathrm{emp}\}$ is provable by (Skip). Then $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp}\}\mathrm{skip}\{\neg\mathrm{emp}\}$ is provable by (Weakening). Therefore both $\Gamma \vdash \{A\}P\{B\}$ and $\Gamma \vdash \{C\}P\{D\}$ are provable.

We will show that $\Gamma \vdash \{A \wedge C\}P\{B \wedge D\}$ is not provable. It is $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp} \wedge \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp} \wedge \neg\mathrm{emp}\}$. We have $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}$ by (Identity). But $\mathrm{emp} \to \mathrm{False}$ is false and hence $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp} \wedge \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp} \wedge \neg\mathrm{emp}\}$ is not provable by (Identity), (Weakening) and (Conseq) rules. Then by Lemma 6.1.5, $\{\{\mathrm{emp} \vee \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp}\}\} \vdash \{\neg\mathrm{emp} \wedge \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp} \wedge \neg\mathrm{emp}\}$ is not provable since $\{\neg\mathrm{emp} \wedge \neg\mathrm{emp}\}\mathrm{skip}\{\mathrm{emp} \wedge \neg\mathrm{emp}\}$ is false.

Therefore, the Conjunction Rule is not admissible. $\square$

This section has revealed the fine structure (and the subtlety) of the problem. The completeness issue studied in this dissertation is much more subtle and more difficult than what one might expect. So, a very careful choice of axioms and inference rules is necessary. The discussion in this section is an evidence of such difficulty in the case of hypothetical judgments.

7 Conclusion

In our work, we have presented a system that can verify all terminating programs written in the language proposed in [18] extended with mutual recursive procedures. Our assertion language is exactly the same as that of [18]. We have shown that our proposed system is sound and relatively complete (in the sense of Cook [8]). The adaptation completeness is straightforward, as the axioms of atomic commands are chosen according to the weakest preconditions. Yet the completeness result could not be achieved from the traditional Hoare's logic for pointer programs simply by choosing the set of appropriate rules. In [3], the expressiveness is assumed and the strongest postcondition is obtained directly from the weakest precondition. In our work, the expressiveness is proved and the precondition for the abort-free execution is established which is necessary to utilize the strongest postcondition.

A future work can be the completeness of the system with non-empty hypothesis. Modification of some axioms and inference rules of this system along with inclusion of some new rules can be a starting point to achieve it. Besides, several extensions of the current system are possible and it is important to study their completeness. Moreover, it is necessary to be able to verify programs written in modern programming languages which are enriched with newer features. Among them, enhancement of procedures that can handle parameters is important. For that it may require to extend the programming language to local variables and parameters. It may pose a challenge to correctly model the local scoping of store and heap. It will also be necessary to handle different types of parameters like call by name, call by value and call by variable. One direction can be the inclusion of the corresponding inference rules from [3]. However, it is necessary to investigate them carefully since not all the sound rules in [3] are consistent in Separation Logic. It is out of the scope of the current work and interesting for the future work.

Including implementation of the system, other future works can be bug tracking in programs and program synthesizing using our system.

References

[1] M.F. Al Ameen. M. Tatsuta, New Complete System of Hoare's Logic with Recursive Procedures, *Constructivism and Computability*, JAIST Logic Workshop Series 2015, Kanazawa, Japan.

[2] M.F. Al Ameen and M. Tatsuta, Completeness for recursive procedures in separation logic, *Theoretical Computer Science*, 2016, Available online 11 April 2016, ISSN 0304-3975, DOI=http://dx.doi.org/10.1016/j.tcs.2016.04.004. (http://www.sciencedirect.com/science/article/pii/S0304397516300329).

[3] K.R. Apt, Ten Years of Hoare's Logic: A Survey — Part I, *ACM Transactions on Programming Languages and Systems* 3 (4) (1981) 431–483.

[4] J. Berdine, C. Calcagno, and P.W. O'Hearn, Symbolic Execution with Separation Logic, In: Proceedings of the Third Asian Symposium on Programming Languages and Systems (APLAS2005), *Lecture Notes in Computer Science* 3780 (2005) 52–68.

[5] J Berdine, C Calcagno and PW O'Hearn, Smallfoot: Modular Automatic Assertion Checking with Separation Logic, In: 4th Formal Methods for Components and Objects, *Lecture Notes in Computer Science* 4111, (2006).

[6] J.A. Bergstra and J.V. Tucker, Expressiveness and the Completeness of Hoare's Logic, *Journal Computer and System Sciences* 25 (3) (1982) 267–284.

[7] C. Calcagno, P.W. O'Hearn, H. Yang. Local Action and Abstract Separation Logic. In: *ACM/IEEE Symposium on Logic in Computer Science (LICS 2007), 2007*.

[8] S.A. Cook, Soundness and completeness of an axiom system for program verification, *SIAM Journal on Computing* 7 (1) (1978) 70–90.

[9] G.A. Gorelick, Complete axiomatic system for proving assertions about recursive and non-recursive programs, *Technical Report No. 75*, Computer Science Dept, University of Toronto, Toronto, Canada, Jan 1975.

[10] J.Y. Halpern, A good Hoare axiom system for an ALGOL-like language, In: *Proceedings of 11th ACM symposium on Principles of programming languages (POPL84)* (1984) 262–271.

[11] C.A.R. Hoare, An axiomatic basis for computer programming, *Commun. ACM* 12 (10) (1969) 576–580,583.

[12] M. Huth, M. Ryan, In: *Logic in Computer Science: Modeling and Reasoning about Systems*, second edition, Cambridge University Press, 2004.

[13] S. Ishtiaq and P.W. O'Hearn, BI as an Assertion Language for Mutable Data Structures, In: *Proceedings of 28th ACM Symposium on Principles of Programming Languages (POPL2001)* (2001) 14–26.

[14] B. Josko, On expressive interpretations of a Hoare-logic for Clarke's language L4, In: Proceedings of 1st Annual Symposium of Theoretical Aspects of Computer Science (STACS 84), *Lecture Notes in Computer Science* 166 (1984) 73–84.

[15] H. H. Nguyen, C. David, S.C. Qin, and W.N. Chin, Automated Verification of Shape and Size Properties Via Separation Logic, In: Proceedings of 8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2007), *Lecture Notes in Computer Science* 4349 (2007) 251–266.

[16] H.H. Nguyen and W.N. Chin, Enhancing Program Verification with Lemmas, In: Proceedings of 20th International Conference on Computer Aided Verification (CAV 2008), *Lecture Notes in Computer Science* 5123 (2008) 355–369.

[17] P.W. O'Hearn, H. Yang, and J.C. Reynolds, Separation and Information Hiding, In: *Proceeding of the 31st Annula Symposium on Principles of Programming Languages (POPL 2004), 2004.*

[18] J.C. Reynolds, Separation Logic: A Logic for Shared Mutable Data Structures, In: *Proceedings of Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS2002)* (2002) 55–74.

[19] M. Tatsuta, W.N. Chin, and M.F. Al Ameen, Completeness of Pointer Program Verification by Separation Logic, In: *Proceeding of 7th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2009)* 179–188.

[20] P.W. O'Hearn and D. J. Pym, The logic of bunched implications In: *Bulletin of Symbolic Logic, 5(2),* June 1999, 215–244

[21] Peter O'Hearn, John Reynolds, Hongseok Yang, Local Reasoning about Programs that Alter Data Structures In: *Proceedings of CSL'01*, LNCS 2142, Paris, 2001. 1–19

[22] Tarski, Alfred. A lattice-theoretical fixpoint theorem and its applications. In: *Pacific J. Math.* 5 (1955), no. 2, 285–309.

[23] H. Yang, Local Reasoning for Stateful Programs, In: *Ph.D. thesis, University of Illinois at Urbana Champaign, 2001*.

Index