EXTENDING HOARE TYPE THEORY FOR ARRAYS

配列のためのホーア型理論の拡張

by

Yuki Watanabe

渡邊　裕貴

Senior Thesis

卒業論文

Submitted to

the Department of Information Science

the Faculty of Science, the University of Tokyo

on February 10, 2009

in Partial Fulfillment of the Requirements

for the Degree of Bachelor of Science

Thesis Supervisor: Akinori Yonezawa 米澤　明憲

Professor of Computer Science

ABSTRACT

Hoare Type Theory is a recently proposed type system for reasoning about programs that contain non-effectful higher-order functions and effectful operations. It integrates Hoare logic into a typed lambda calculus by encapsulating effectful operations as monads of the type of Hoare triples, which specifies the precondition, the type of the result value, and the postcondition of operations. However, in previous work, the number of updated locations is defined in the assertion statically; therefore, we cannot update multiple locations if the number of the locations is given at runtime. In this thesis, Hoare Type Theory is extended to allow typing such programs, by changing the notion of update in the assertion logic. The new system can type programs that deal with adjacent multiple locations of runtime-defined length, which can be considered as arrays.

論文要旨

ホーア型理論 (Hoare Type Theory) は、副作用のない高階関数と副作用のある処理とを含むプログラムの正当性を論証するための、近年提案された型システムである。この論理では副作用のある処理をモナドとして扱うことでホーア論理を型付きラムダ計算に組み込んでおり、それらのモナドの型はその処理の事前条件・結果の型・事後条件を示すホーア式となっている。しかしこれまでの提案では、プログラム内で更新されるロケーションの数は表明によって静的に決められており、ロケーションの数が動的に変化する場合を扱うことはできなかった。この論文では、更新されるロケーションの数が動的に決められるように、表明におけるロケーションの更新の扱いが変更される。新しい型システムでは、動的に数が決まる連続したロケーションを配列として扱えるようになる。

ACKNOWLEDGEMENT

I would like to thank Prof. A. Yonezawa and Asst. Prof. T. Maeda for useful advice.

CONTENTS

TABLE OF FIGURES

1 Introduction

Functional programming and imperative programming are two of the most important programming paradigms.

In functional programming, computation is performed by evaluating functions and treatment of states is intentionally avoided. Because evaluation of a function does not depend on or affect states, any function evaluates to the same value for the same argument. This property is called referential transparency and it establishes independence between functions. Thus, when we check correctness of a program, we do not have to pay attention to relationships or interactions between functions in the program; check of correctness of the whole program can be done by simply checking each of the functions in the program.

Lambda calculus is a popular system that embodies basic ideas of the functional programming paradigm. The lambda calculus that contains formal definitions for type checking is called typed lambda calculus. A variety of typed lambda calculi has been studied for decades. The most basic one is the simply typed lambda calculus [5], which gives simple types of functions to the untyped lambda calculus. System F [7, 19] adds polymorphism to the simply typed lambda calculus by using universal quantification over types. Dependent types [6, 8] are types that depend on values, which can be considered attributes that qualify the types.

In imperative programming, on the other hand, computation is mainly performed by executing statements that change the state. A change of a state is called a side effect or simply an effect. The result of execution of a statement depends on the state, so conditions about the state between each statement must be examined to check correctness of a program.

Hoare logic [9] is a classic system for reasoning about imperative programs. The main feature of Hoare logic is the Hoare triple, which comprises a precondition, a postcondition and a statement. A Hoare triple states that if the precondition is satisfied before the statement is executed and if the execution terminates, then the postcondition is satisfied after the execution. In Hoare logic, correctness of a program is checked by ensuring consistency of all the pre- and postconditions, which are expressed as logical propositions and are called assertions. Part of the logic that formally defines the way to ensure consistency of assertions is called the assertion logic.

Hoare Type Theory (HTT) [14, 15, 16] is a type theory which integrates Hoare logic into a dependently typed lambda calculus. In HTT, effectful computations are encapsulated into monads [13, 21, 22] so that they can appear in non-effectful terms. Those monads have the type of Hoare triples, which is made up of the precondition, the postcondition and the type of the result of the effectful computation. We can combine higher-order functions from the lambda calculus with Turing-complete imperative programs and check correctness of the combined programs using the assertion logic, whereas higher-order functions typically cannot appear in Hoare logic.

Use of HTT for practical applications, however, has not been studied well. Mutable arrays, for instance, are one of the most basic data structures that are used in imperative programming, but HTT cannot effectively handle arrays.

One reason for this is that it lacks the way to allocate arrays. Allocation of arrays is difficult because the *alloc* command, which is the primitive computation that allocates a heap location, is not defined to allocate multiple locations. We need a way to allocate multiple adjacent locations, which can be used as an array.

Another reason comes from the limitation in the assertion logic. In the assertion logic of the original HTT [14], the notion of heap updates is represented by the update construct adapted from [4]; each instance of updates is represented by a corresponding instance of the update construct. Although multiple updates to the same location can be contracted to one invocation of the update construct, those to different locations cannot. Therefore, the number of updated locations in a program is determined statically as specified in the assertion for the entire program, making it impossible to type-check such a program that the number of updated locations is determined at runtime.

In this thesis, the original HTT is extended in order to make it possible to handle arrays by changing the semantics of the *alloc* command and redefining treatment of heaps in the asser-

Types	A, B, C	$::=$	$\alpha \mid \mathrm{bool} \mid \mathrm{nat} \mid \mathrm{unit} \mid \Pi x{:}A.B \mid \{P\}x{:}A\{Q\}$
Primitive propositions	p	$::=$	$\top \mid \bot \mid \mathrm{id}_A(M, N) \mid \mathrm{seleq}_A(h, M, N)$
Propositions	P, Q, R, I	$::=$	$p \mid P \wedge Q \mid P \vee Q \mid P \supset Q \mid \neg P \mid$
			$\forall x{:}A.P \mid \exists x{:}A.P \mid$
			$\forall \alpha{:}\mathrm{type}.P \mid \exists \alpha{:}\mathrm{type}.P \mid \forall h{:}\mathrm{heap}.P \mid \exists h{:}\mathrm{heap}.P$
Elimination terms	K, L	$::=$	$x \mid K\ M \mid M : A$
Introduction terms	M, N, O	$::=$	$K \mid () \mid \lambda x.M \mid \mathrm{dia}\ E \mid \mathrm{true} \mid \mathrm{false} \mid$
			$\mathrm{zero} \mid \mathrm{succ}\ M \mid M + N \mid M \times N \mid$
			$\mathrm{eq}(M, N) \mid \mathrm{le}(M, N) \mid \mathrm{lt}(M, N)$
Commands	c	$::=$	$x = \mathrm{alloc}\ M \mid x = [M]_A \mid [M]_A = N \mid$
			$x = \mathrm{if}_A(M, E_1, E_2) \mid x = \mathrm{loop}_A^I(M, y.N, y.F) \mid$
			$x = \mathrm{fix}_A(f.y.F, M)$
Computations	E, F	$::=$	$M \mid \mathrm{let\ dia}\ x = K\ \mathrm{in}\ E \mid c; E$
Variable contexts	Δ	$::=$	$\cdot \mid \Delta, x{:}A \mid \Delta, \alpha$
Heap contexts	Ψ	$::=$	$\cdot \mid \Psi, h$
Proposition contexts	Γ	$::=$	$\cdot \mid \Gamma, P$

Here α represents a type variable and h represents a heap variable.

Figure 1: Syntax of Hoare Type Theory

tion logic. Changes in this thesis can be summarized as follows:

- The update construct is no longer used in the assertion logic and assertions about heaps are expressed by propositions that relate states of each location between effects.
- The changed *alloc* command allocates multiple adjacent locations whose exact number is not known at the type checking.
- The terms are enriched with the new *le* and *lt* comparison operators (standing for "less-equal" and "less-than" respectively), which are not formally defined in the previous work [14, 15, 16] but useful for judgment of termination of a loop.

The rest of this thesis is organized as follows. In Section 2, the syntax and its intuitive semantics of the extended HTT are described. Section 3 describes the judgments used in type checking. Section 4 shows lemmas and theorems on the type system. Section 5 defines the operational semantics of HTT and shows soundness of the type system. Section 6 gives two example programs that deal with arrays. Section 7 discusses related and future work about HTT. Finally, Section 8 concludes the thesis.

2 Syntax

The syntax of HTT is defined as in Figure 1, following [14] with a few changes.

The primitive types of HTT consist of natural numbers, Booleans and unit. HTT has no location type because natural numbers are used to represent heap locations. The other types are the dependent function type and the computation type. In the computation type $\{P\}x{:}\,A\{Q\}$, P is the precondition, A is the type of the result of the computation, and Q is the postcondition, where A and Q may depend on the variable x, which represents the result value. The precondition is a condition which must be satisfied to ensure the computation does not get stuck. The postcondition is a condition which is always satisfied after the computation terminates.

The primitive propositions include the *id* and *seleq* predicates. The proposition $\mathrm{id}_A(M, N)$ asserts equality of two terms M and N of type A. The proposition $\mathrm{seleq}_A(h, M, N)$ asserts that the location M of the heap h contains the term N of type A. Non-primitive propositions may contain universal and existential quantifications over types, values and heaps. Quantification on types is adopted from [15] to allow asserting about locations whose content types and values are unknown. To this end, types and variable contexts are also extended with type variables. However, the type variables are not used to achieve polymorphism in this thesis because it is beyond the scope of this thesis.

The following notations are defined to simplify propositions, in the same way as in [15]:

$$
\begin{array}{ll}
M \in h & \coloneqq \exists\alpha{:}\,\mathrm{type}.\, \exists v{:}\,\alpha.\, \mathrm{seleq}_\alpha(h, M, v) \\
M \notin h & \coloneqq \neg(M \in h) \\
P \Leftrightarrow Q & \coloneqq (P \supset Q) \wedge (Q \supset P) \\
\mathrm{seleq}_A(h, M, -) & \coloneqq \exists x{:}\,A.\, \mathrm{seleq}_A(h, M, x) \\
\mathrm{share}(h_1, h_2, M) & \coloneqq \forall\alpha{:}\,\mathrm{type}.\, \forall v{:}\,\alpha.\, \mathrm{seleq}_\alpha(h_1, M, v) \Leftrightarrow \mathrm{seleq}_\alpha(h_2, M, v) \\
\mathrm{hid}(h_1, h_2) & \coloneqq \forall x{:}\,\mathrm{nat}.\, \mathrm{share}(h_1, h_2, x)
\end{array}
$$

$M \in h$ denotes that the location M is allocated in the heap h. $\mathrm{share}(h_1, h_2, M)$ denotes that the content of the location M is the same between the two heaps h_1 and h_2. $\mathrm{hid}(h_1, h_2)$ denotes that the two heaps are identical.

A heap, semantically, is a finite partial function which maps a natural number to a pair of a value and its type. Syntactically, however, heaps only appear in the form of heap variables, which each denote the whole state of a heap at a particular moment between effects. This is one of the differences from [14], which had a special construct (*upd*) to denote update of a heap. In this thesis, such a construct is not used. Instead, the *seleq* predicate, adopted from [15], is defined as a primitive proposition and is used to assert the content of a location at a particular moment.

Terms include primitive values such as true and (), and some arithmetic operations. The comparison operations *eq*, *lt* and *le* evaluates to true or false according to whether their first operand is equal to, less than and either equal to or less than their second operand respectively. The term $\mathrm{dia}\, E$ is an encapsulated computation. Terms are divided into two categories, i.e. elimination terms (or elim terms) and introduction terms (or intro terms), which correspond to the two forms of the typing judgments for terms. This approach is due to Watkins et al [23].

Computations are the effectful part of HTT. Computations that can be separated by semicolons are especially called commands.

- The command $x = \mathrm{alloc}\, M$ allocates adjacent M heap locations and binds the variable x to the first location.
- $x = [M]_A$ looks up the content of the location M and $[M]_A = N$ updates the content of the location to N, where A is the type of the content value.
- The conditional command $x = \mathrm{if}_A(M, E_1, E_2)$ binds x to the result of the computation E_1 or E_2 according to the value of the Boolean term M.
- The loop command $x = \mathrm{loop}_A^I(M, y.N, y.F)$ executes the computation F while the Boolean term N evaluates to true. Each time before N is evaluated or F is computed, the free variable y in N or F is replaced with the loop counter. The initial loop counter is M and the result of F becomes the next loop counter after each iteration. When N evaluates to false, x is bound to the then loop counter. The proposition I expresses the loop invariant which denotes the effect of the loop and the type A is the type of the loop counter,

which is also the type of the result of F.

- The fixpoint command $x = \mathrm{fix}_A(f.y.F, M)$ applies the fixpoint of the equation $f = \lambda y.\,\mathrm{dia}\,F$ to M and binds x to the result of it. The type A is the type of the fixpoint.
- In the computation $\mathrm{let\,dia}\,x = K\ \mathrm{in}\ E$, the term K must evaluate to an encapsulated computation, which is subsequently executed. The result of the computation is bound to the variable x and then the computation E is executed.

3 Type System

The basic structure of the type system is described in this section, though most part of it remains the same as in [14]. The full definition of the type system is presented in the appendix.

The type system contains several kinds of judgments. Some judgments synthesize the *canonical form* of the expression on which the judgment is performed. The canonical form is the semantically equivalent expression which is in the normal and eta-long form, that is, beta reduction and eta expansion are fully performed on all its subterms. The intention of introducing the canonical form is that terms that are syntactically different but semantically equivalent are canonicalized to the same form (except for alpha-equivalence), so that in the assertion logic semantic equivalence of terms can be checked by simply comparing the syntactic structures of their canonical forms (after proper alpha-conversions). In the judgments, the synthesized canonical forms are given in brackets and usually denoted by primed variables.

Judgments on well-formedness of expressions are in the following forms:

$$\vdash \Delta\ \mathrm{ctx}$$

$$\Delta \vdash A \Leftarrow \mathrm{type}\ [A']$$

$$\Delta \vdash P \Leftarrow \mathrm{prop}\ [P']$$

The type checking judgments have the following forms:

$$\Delta \vdash K \Rightarrow A\ [N']$$

$$\Delta \vdash M \Leftarrow A\ [M']$$

$$\Delta; P \vdash E \Rightarrow x{:}A.Q\ [E']$$

$$\Delta; P \vdash E \Leftarrow x{:}A.Q\ [E']$$

The other judgment is the sequent, which is used in the assertion logic:

$$\Delta; \Psi; \Gamma_1 \vdash \Gamma_2$$

3.1 Hereditary and monadic substitutions

The process of canonicalization of a term includes repeated beta reduction of the subterms. To achieve this, a special substitution operation is used, which is called *hereditary substitution*. Hereditary substitution differs from the normal substitution operation in that if substitution makes a new redex it is immediately reduced by recursively applying hereditary substitution. To ensure that hereditary substitution terminates despite recursion, the type of the substituted term is used as a metric. If the metric does not satisfy the specific conditions, hereditary substitution fails, and so does the whole type checking.

A hereditary substitution is denoted as $[x \mapsto M]^*_S(X)$, where S is the type of M which is used as the metric, and X is an expression in which substitution is performed. * is one of a, k, m, e, and p, and specifies the syntactic category of X: a for types, k for elim terms, m for intro terms, e for computations, and p for propositions. Precisely, S is not a type, but the *shape* of a type, which can be taken as an abstract structure of the type. Using the shape of types (rather than types themselves) as the metric makes simpler the definition of and proofs about hereditary substitutions. The shape of a type A is formally denoted as A^-, but is often written simply as A if not ambiguous.

While testing the metrics in the substitution, it is required to check if a shape is a syntactic subexpression of another shape. This check can obviously be done inductively in finite time. It is denoted as $S_1 \preccurlyeq S_2$ that S_1 is a subexpression of S_2, and $S_1 \prec S_2$ that S_1 is a proper subexpression of S_2.

As beta reduction on terms is calculated by term substitution, beta reduction on computations is calculated by a proper substitution operation adopted from [18], which is called *monadic substitution*. The monadic substitution $\langle x{:}A \mapsto E\rangle F$ denotes substitution of the free variable x in the computation F with the result of computation E. Monadic substitution also has a hereditary version of it, which is denoted as $\langle x \mapsto E\rangle_S F$.

The rest of this subsection shows some properties of hereditary substitutions, following [14].

Theorem 1: Termination of hereditary substitutions

1. If $[x \mapsto M]^{\mathrm{k}}_S(K) = N' :: S_1$, then $S_1 \preccurlyeq S$.
2. $[x \mapsto M]^{*}_S(X)$ and $\langle x \mapsto E\rangle_S(F)$ terminate in finite time, either in success or in failure.

Proof: The first statement is by induction on K. The second is by nested induction on the structure of S and on that of X. The cases for the newly added operator terms $\mathrm{le}(M, N)$ and $\mathrm{lt}(M, N)$ are analogous to that of $\mathrm{eq}(M, N)$.

Lemma 2: Hereditary substitutions and heads

If $[x \mapsto M]^{\mathrm{k}}_S(K)$ exists, then it is an elim term K' iff $\mathrm{head}(K) \neq x$ and otherwise it is an intro term $M' :: S'$, where the *head* of an elim term is defined as:

$$\begin{aligned} \mathrm{head}(x) &= x \\ \mathrm{head}(K\ N) &= \mathrm{head}(K) \end{aligned}$$

Proof: By induction on the structure of K.

Lemma 3: Trivial hereditary substitutions

If $x \notin \mathrm{FV}(X)$, then $[x \mapsto M]^{*}_S(X) = X$.

Proof: By induction on the structure of X.

Lemma 4: Hereditary substitutions and primitive operations

Assuming that $[x \mapsto M]^{\mathrm{m}}_S(N_1)$ and $[x \mapsto M]^{\mathrm{m}}_S(N_2)$ exist, the following equations hold:

1. $[x \mapsto M]^{\mathrm{m}}_S(\mathrm{plus}(N_1, N_2)) = \mathrm{plus}([x \mapsto M]^{\mathrm{m}}_S(N_1), [x \mapsto M]^{\mathrm{m}}_S(N_2))$
2. $[x \mapsto M]^{\mathrm{m}}_S(\mathrm{times}(N_1, N_2)) = \mathrm{times}([x \mapsto M]^{\mathrm{m}}_S(N_1), [x \mapsto M]^{\mathrm{m}}_S(N_2))$
3. $[x \mapsto M]^{\mathrm{m}}_S(\mathrm{equals}(N_1, N_2)) = \mathrm{equals}([x \mapsto M]^{\mathrm{m}}_S(N_1), [x \mapsto M]^{\mathrm{m}}_S(N_2))$
4. $[x \mapsto M]^{\mathrm{m}}_S(\mathrm{lessequal}(N_1, N_2)) = \mathrm{lessequal}([x \mapsto M]^{\mathrm{m}}_S(N_1), [x \mapsto M]^{\mathrm{m}}_S(N_2))$
5. $[x \mapsto M]^{\mathrm{m}}_S(\mathrm{lessthan}(N_1, N_2)) = \mathrm{lessthan}([x \mapsto M]^{\mathrm{m}}_S(N_1), [x \mapsto M]^{\mathrm{m}}_S(N_2))$

Proof: By induction on the structure of N_1 and N_2.

Lemma 5: Composition of hereditary substitutions

1. If $y \notin \mathrm{FV}(M_0)$, and $[x \mapsto M_0]^{*}_A(X) = X_0$, $[y \mapsto M_1]^{*}_B(X) = X_1$ and $[x \mapsto M_0]^{\mathrm{m}}_A(M_1)$ exist,

then $[x \mapsto M_0]^*_A(X_1) = [y \mapsto [x \mapsto M_0]^{\mathrm{m}}_A(M_1)]^*_B(X_0)$.

2. If $y \notin \mathrm{FV}(M_0)$, and $[x \mapsto M_0]^{\mathrm{e}}_A(F) = F_0$, $\langle y \mapsto E_1\rangle_B(F) = F_1$ and $[x \mapsto M_0]^{\mathrm{e}}_A(E_1)$ exist,

   then $[x \mapsto M_0]^{\mathrm{e}}_A(F_1) = \langle y \mapsto [x \mapsto M_0]^{\mathrm{e}}_A(E_1)\rangle_B(F_0)$.

3. If $x \notin \mathrm{FV}(F)$, and $\langle y \mapsto E_1\rangle_B(F) = F_1$and $\langle x \mapsto E_0\rangle_A(E_1)$ exist,

   then $\langle x \mapsto E_0\rangle_A(F_1) = \langle y \mapsto \langle x \mapsto E_0\rangle_A(E_1)\rangle_B(F)$.

Proof: By nested induction, first on the shapes A^- and B^-, then on the structure of the expressions in which substitution is performed.

3.2 Terms

The type checking judgment for intro terms has the form $\Delta \vdash K \Rightarrow A\ [N']$ and infers the type A of the term K. The judgment for elim terms has the form $\Delta \vdash M \Leftarrow A\ [M']$ and checks the term K against the given type A.

The auxiliary functions *plus*, *times*, *equals*, *lessequal* and *lessthan* are used to compute canonical forms of corresponding primitive operation terms. The last two are the ones which were newly added in this thesis and have definitions analogous to that of *equals*. They are used in canonicalization of the newly added terms *le* and *lt*.

Since locations in HTT are just natural numbers, the capability of natural number arithmetic directly defines the capability of pointer arithmetic. The natural number arithmetic of HTT includes addition, multiplication, and comparison, which are mostly enough for pointer arithmetic to deal with arrays. The arithmetic is also used in the assertion logic through canonicalization of terms.

3.3 Computations

The judgments for computations also have two forms: $\Delta; P \vdash E \Rightarrow x{:}A.Q\ [E']$ and $\Delta; P \vdash E \Leftarrow x{:}A.Q\ [E']$. The former checks that the result type of the computation E is A and calculates the strongest postcondition Q from the precondition P. The latter checks if Q is a valid postcondition for the computation E as well as if A is a valid result type of E. The assertion logic, which is described below, is used for the latter. That is, it is checked if the calculated strongest postcondition implies the given postcondition Q. In both the judgments, A and Q may depend on the variable x, which denotes the result of the computation.

The special heap variables *mem* and *init* are used in pre- and postconditions. In preconditions, *mem* denotes the heap just before the computation. In postconditions, *init* denotes the heap just before the computation and *mem* the heap just after the computation.

As the notion of heaps is changed in this thesis, the strongest postconditions calculated by the typing rules are changed accordingly. For instance, the strongest postcondition of update is defined as:

$$\mathrm{sp}([M]_A = N) \coloneqq \mathrm{seleq}_A(\mathrm{mem}, M, N) \wedge \forall n{:}\, \mathrm{nat}.\, \neg\mathrm{id}_{\mathrm{nat}}(n, M) \supset \mathrm{share}(\mathrm{init}, \mathrm{mem}, n).$$

This states that the location M contains the term N of type A after the update and that any other location is not changed in the update.

The major change in the definition of the computations from [14] is the definition of the *alloc* command. In [14], this command allocates a new heap location and initializes it with the given argument. Since an arbitrary unused location is selected for the new location, one cannot assume any ordering or continuity between locations allocated by multiple invocations of the command. Therefore, one cannot allocate adjacent locations, which we want to use as an array. In this thesis, on the other hand, the *alloc* command allocates multiple adjacent locations at a time and the number of the locations can be specified by the argument of the natural number type. This allows allocation of an array whose length is statically unknown. The contents of the newly allocated locations are initialized to the unit value. Since strong update is possible in this type system, the initial values and their types are not important.

The strongest postcondition calculated by the type checking for the *alloc* command is defined as:

$$\begin{aligned}\mathrm{sp}(x = \mathrm{alloc}\, M) \;\coloneqq\; & \forall n{:}\, \mathrm{nat}.\, \big(n < x \vee x + M \leq n \supset \mathrm{share}(\mathrm{init}, \mathrm{mem}, n)\big) \wedge \\ & \big(x \leq n \wedge n < x + M \supset \neg\mathrm{indom}(\mathrm{init}, n) \wedge \mathrm{seleq}_{\mathrm{unit}}(\mathrm{mem}, n, ())\big).\end{aligned}$$

This states that M locations, the first of which is x, are allocated and initialized to the unit value. The proposition $\mathrm{share}(\mathrm{init}, \mathrm{mem}, n)$ states that the location n is not changed by the *alloc* command, where n is a location that is not any of the locations allocated. The proposition

$\neg\mathrm{indom}(\mathrm{init}, n) \wedge \mathrm{seleq}_{\mathrm{unit}}(\mathrm{mem}, n, ())$ states that the location n is not allocated before the command and that the location contains the unit value after the command, where n is one of the locations allocated by the command.

3.4 Assertion logic

The assertion logic is a simple sequent calculus. Some derivation rules about heaps which were in [14] are removed in this thesis because they are no longer needed now that heap expressions are not used in the assertion logic. Instead, universal and existential quantifications on types are adopted from [15]. Universal quantification on types is essential for the definition of the *share* expression because the expression is used to relate arbitrary locations, the type of whose content is unknown.

Inequality of natural numbers can be expressed as the following in the assertion logic:

$$M \leq N \coloneqq \exists x{:}\, \mathrm{nat}.\, \mathrm{id}_{\mathrm{nat}}(M + x, N)$$

$$M < N \coloneqq \mathrm{succ}\, M \leq N$$

Some mathematical properties about inequality are shown in the next section.

4 Properties

This section shows basic properties of HTT. Most of the proofs for these properties can be done in the same manner as in [14], and thus are not given in detail. Especially, in case analysis, cases for the newly added operator terms *le* and *lt* are analogous to that of the term *eq*.

At the last of this section are basic mathematical properties, which are not mentioned in [14], but should be worth mentioning to convince the reader that the logic is strong enough to validate the example programs shown in Section 6.

Theorem 6: Relative decidability of type checking

If validity of every assertion sequent is decidable, then all the typing judgments of HTT are decidable.

Proof: By induction on the structure of the typing judgment.

As stated in the theorem, decidability of the whole type system depends on the assumption that the assertion logic is decidable. Decidability of the assertion logic is still not established in this thesis.

Lemma 7: Context weakening and contraction

Let $\Delta \vdash J$ be a judgment of HTT that depends on a variable context Δ, and $\Delta; \Psi \vdash J$ be a judgment that depends on a variable context Δ and a heap context Ψ.

1. If $\Delta \vdash J$ and $\Delta \vdash A \Leftarrow \text{type}\ [A]$, then $\Delta, x{:}\, A \vdash J$.
2. If $\Delta; \Psi \vdash J$, then $\Delta; \Psi, h \vdash J$.
3. If $\Delta, x{:}\, A, \Delta_1, y{:}\, A, \Delta_2 \vdash J$, then $\Delta, x{:}\, A, \Delta_1, [y \mapsto x]\Delta_2 \vdash [y \mapsto x]J$.
4. If $\Delta; \Psi, h, \Psi_1, g, \Psi_2 \vdash J$, then $\Delta; \Psi, h, \Psi_1, \Psi_2 \vdash [g \mapsto h]J$.

Proof: By straightforward induction on the derivation of *J*.

Lemma 8: Closed canonical forms of primitive types

1. If $\vdash M \Leftarrow \text{nat}\ [M]$, then $M = \text{succ}^n\ \text{zero}$ for some natural number *n*.
2. If $\vdash M \Leftarrow \text{bool}\ [M]$, then $M = \text{true}$ or $M = \text{false}$.

Proof: By induction of the structure of *M*.

Lemma 9: Properties of variable expansion

1. If $\Delta, x{:}\, A, \Delta_1 \vdash K \Rightarrow B\ [K]$, then $[x \mapsto \text{expand}_A(x)]^{\text{k}}_A(K)$ exists, and
   (a) if $[x \mapsto \text{expand}_A(x)]^{\text{k}}_A(K) = K'$, then $K' = K$
   (b) if $[x \mapsto \text{expand}_A(x)]^{\text{k}}_A(K) = N' :: S$, then $N' = \text{expand}_B(K)$ and $S = B^-$.
2. If $\Delta, x{:}\, A, \Delta_1 \vdash N \Leftarrow B\ [N]$, then $[x \mapsto \text{expand}_A(x)]^{\text{m}}_A(N) = N$.
3. If $\Delta, x{:}\, A, \Delta_1; P \vdash E \Leftarrow y{:}\, B.\, Q\ [E]$, then $[x \mapsto \text{expand}_A(x)]^{\text{e}}_A(E) = E$.
4. If $\Delta, x{:}\, A, \Delta_1 \vdash B \Leftarrow \text{type}\ [B]$, then $[x \mapsto \text{expand}_A(x)]^{\text{a}}_A(B) = B$.
5. If $\Delta, x{:}\, A, \Delta_1; \Psi \vdash P \Leftarrow \text{prop}\ [P]$, then $[x \mapsto \text{expand}_A(x)]^{\text{p}}_A(P) = P$.
6. If $\Delta \vdash M \Leftarrow A\ [M]$, then $[x \mapsto M]^{\text{m}}_A\big(\text{expand}_A(x)\big) = M$.
7. If $\Delta; P \vdash E \Leftarrow x{:}\, A.\, Q\ [E]$, then $\langle x \mapsto E\rangle_A\big(\text{expand}_A(x)\big) = E$.

Proof: By mutual nested induction on the structure of the type A and the substituted expression.

Lemma 10: Identity principles

1. If $\Delta; \Psi \vdash P \Leftarrow \text{prop}\ [P]$, then $\Delta; \Psi; \Gamma_1, P \vdash P, \Gamma_2$.

2. If $\Delta; \Psi; \Gamma_1, \mathrm{id}_B(M, N) \vdash [x \mapsto M]^{\mathrm{p}}_B(P), \Gamma_2$ and $[x \mapsto N]^{\mathrm{p}}_B(P)$ is well-formed and canonical, then $\Delta; \Psi; \Gamma_1, \mathrm{id}_B(M, N) \vdash [x \mapsto N]^{\mathrm{p}}_B(P), \Gamma_2$.
3. If $\Delta; \Psi; \Gamma_1, \mathrm{hid}(h_1, h_2) \vdash [h \mapsto h_1]P, \Gamma_2$, then $\Delta; \Psi; \Gamma_1, \mathrm{hid}(h_1, h_2) \vdash [h \mapsto h_2]P, \Gamma_2$.
4. If $\Delta \vdash K \Rightarrow A\ [K]$, then $\Delta \vdash \mathrm{expand}_A(K) \Leftarrow A\ [\mathrm{expand}_A(K)]$.

Proof: By simultaneous induction on the structures of P and A. The difference from [14] is that *seleq*, not *hid*, is the primitive proposition that asserts on heaps. The case analysis for the statement 3 is slightly changed accordingly. Instead of proofing the case when $P = \mathrm{hid}(h, h')$, we need to prove the case when $P = \mathrm{seleq}_A(h, M, N)$. This case is easily shown by the definition of $\mathrm{hid}(h_1, h_2)$.

Lemma 11: Properties of computations

If $\Delta; P \vdash E \Leftarrow x{:}A.Q\ [E']$ and

1. if $\Delta; x{:}A; \mathrm{init}, \mathrm{mem}; Q \vdash R$, then $\Delta; P \vdash E \Leftarrow x{:}A.R\ [E']$.
2. if $\Delta; \mathrm{init}, \mathrm{mem}; R \vdash P$, then $\Delta; R \vdash E \Leftarrow x{:}A.Q\ [E']$.
3. if $\Delta; \mathrm{init}, \mathrm{mem} \vdash R \Leftarrow \mathrm{prop}\ [R]$, then $\Delta; R \circ P \vdash E \Leftarrow x{:}A.(R \circ Q)[E']$.

Proof: The first statement is by simple deduction using the derivation rule for $\Delta; P \vdash E \Leftarrow x{:}A.Q\ [E']$ and the cut rule. The second and third are by induction on the structure of E.

Lemma 12: Canonical substitution principles

Assuming $\Delta \vdash M \Leftarrow A\ [M]$ and $\vdash (\Delta, x{:}A, \Delta_1)\mathrm{ctx}$ and that the context $\Delta_1' = [x \mapsto M]_A(\Delta_1)$ exists and is well-formed (i.e. $\vdash (\Delta, \Delta_1')$ ctx), the following statements hold:

1. If $\Delta, x{:}A.\Delta_1 \vdash K \Rightarrow B\ [K]$, then $[x \mapsto M]^{\mathrm{k}}_A(K)$ and $B' = [x \mapsto M]^{\mathrm{a}}_A(B)$ exist, B' is well-formed (i.e. $\Delta, \Delta_1' \vdash B' \Leftarrow \mathrm{type}\ [B']$) and
   (a) if $[x \mapsto M]^{\mathrm{k}}_A(K) = K'$, then $\Delta, \Delta_1' \vdash K' \Leftarrow B'\ [K']$
   (b) if $[x \mapsto M]^{\mathrm{k}}_A(K) = N' :: S$, then $\Delta, \Delta_1' \vdash N' \Leftarrow B'\ [N']$ and $S = B^-$.
2. If $\Delta, x{:}A, \Delta_1 \vdash N \Leftarrow B\ [N]$ and if the type $B' = [x \mapsto M]^{\mathrm{a}}_A(B)$ exists and is well-formed (i.e. $\Delta, \Delta_1' \vdash B' \Leftarrow \mathrm{type}\ [B']$), then $\Delta, \Delta_1' \vdash [x \mapsto M]^{\mathrm{m}}_A(N) \Leftarrow B'\ [[x \mapsto M]^{\mathrm{m}}_A(N)]$.
3. If $\Delta, x{:}A, \Delta_1; P \vdash E \Leftarrow y{:}B.Q\ [E]$ and $y \notin \mathrm{FV}(M)$ and if the propositions $P' = [x \mapsto M]^{\mathrm{p}}_A(P)$ and $Q' = [x \mapsto M]^{\mathrm{p}}_A(Q)$ and the type $B' = [x \mapsto M]^{\mathrm{a}}_A(B)$ exist and are well-formed,

then $\Delta, \Delta_1'; P' \vdash [x \mapsto M]^{\mathrm{e}}_A(E) \Leftarrow y{:}\, B'.\, Q'\ [[x \mapsto M]^{\mathrm{e}}_A(E)]$.

4. If $\Delta, x{:}\, A, \Delta_1 \vdash B \Leftarrow \text{type}\ [B]$, then $\Delta, \Delta_1' \vdash [x \mapsto M]^{\mathrm{a}}_A(B) \Leftarrow \text{type}\ [[x \mapsto M]^{\mathrm{a}}_A(B)]$.

5. If $\Delta, x{:}\, A, \Delta_1; \Psi \vdash P \Leftarrow \text{prop}\ [P]$, then $\Delta, \Delta_1'; \Psi \vdash [x \mapsto M]^{\mathrm{p}}_A(P) \Leftarrow \text{prop}\ \big[[x \mapsto M]^{\mathrm{p}}_A(P)\big]$.

6. If $\Delta, x{:}\, A, \Delta_1; \Psi; \Gamma_1 \vdash \Gamma_2$ and the proposition context $\Gamma_1' = [x \mapsto M]_A(\Gamma_1)$ and $\Gamma_2' = [x \mapsto M]_A(\Gamma_2)$ exist and are well-formed, then $\Delta, \Delta_1'; \Psi; \Gamma_1' \vdash \Gamma_2'$.

7. If $\Delta; P \vdash E \Leftarrow x{:}\, A.\, Q\ [E]$ and $\Delta, x{:}\, A; Q \vdash F \Leftarrow y{:}\, B.\, R\ [F]$ where $x \notin \mathrm{FV}(B) \cup \mathrm{FV}(R)$, then $\Delta; P \vdash \langle x \mapsto E\rangle_A(F) \Leftarrow y{:}\, B.\, R\ [\langle x \mapsto E\rangle_A(F)]$.

Proof: By nested induction, first on the structure of the shape of A, and next on the derivation of the typing or sequent judgment in each case.

Lemma 13: Idempotence of canonicalization

1. If $\Delta \vdash K \Rightarrow A\ [K']$ where K' is an elim term, then $\Delta \vdash K' \Rightarrow A\ [K']$.
2. If $\Delta \vdash K \Rightarrow A\ [N']$ where N' is an intro term, then $\Delta \vdash N' \Leftarrow A\ [N']$.
3. If $\Delta \vdash N \Leftarrow A\ [N']$, then $\Delta \vdash N' \Leftarrow A\ [N']$.
4. If $\Delta; P \vdash E \Leftarrow x{:}\, A.\, Q\ [E']$, then $\Delta; P \vdash E' \Leftarrow x{:}\, A.\, Q\ [E']$.
5. If $\Delta \vdash A \Leftarrow \text{type}\ [A']$, then $\Delta \vdash A' \Leftarrow \text{type}\ [A']$.
6. If $\Delta; \Psi \vdash P \Leftarrow \text{prop}\ [P']$, then $\Delta; \Psi \vdash P' \Leftarrow \text{prop}\ [P']$.

Proof: By induction on the structure of the typing derivations.

Lemma 14: General substitution principles

Assuming $\Delta \vdash A \Leftarrow \text{type}\ [A']$ and $\Delta \vdash M \Leftarrow A'\ [M']$, the following statements hold:

1. If $\Delta, x{:}\, A', \Delta_1 \vdash K \Rightarrow B\ [N']$,

   then $\Delta, [x \mapsto M']_A(\Delta_1) \vdash [x \mapsto M{:}\, A]K \Rightarrow [x \mapsto M']^{\mathrm{a}}_A(B)\ [[x \mapsto M']^{\mathrm{m}}_A(N')]$.

2. If $\Delta, x{:}\, A', \Delta_1 \vdash N \Leftarrow B\ [N']$,

   then $\Delta, [x \mapsto M']_A(\Delta_1) \vdash [x \mapsto M{:}\, A]N \Leftarrow [x \mapsto M']^{\mathrm{a}}_A(B)\ [[x \mapsto M']^{\mathrm{m}}_A(N')]$.

3. If $\Delta, x{:}\, A', \Delta_1; P \vdash E \Leftarrow y{:}\, B.\, Q\ [E']$ and $y \notin \mathrm{FV}(M)$, then $\Delta, [x \mapsto M']_A(\Delta_1); [x \mapsto M']^{\mathrm{p}}_A(P) \vdash [x \mapsto M{:}\, A]E \Leftarrow y{:}\, [x \mapsto M']^{\mathrm{a}}_A(B).\, [x \mapsto M']^{\mathrm{p}}_A(Q)\ [[x \mapsto M']^{\mathrm{e}}_A(E')]$.

4. If $\Delta, x{:}\, A', \Delta_1 \vdash B \Leftarrow \text{type}\ [B']$,

   then $\Delta, [x \mapsto M']_A(\Delta_1) \vdash [x \mapsto M{:}\, A]B \Leftarrow \text{type}\ [[x \mapsto M']^{\mathrm{a}}_A(B')]$.

5. If $\Delta, x{:}A', \Delta_1; \Psi \vdash P \Leftarrow \text{prop}\ [P']$,

   then $\Delta, [x \mapsto M']_A(\Delta_1); \Psi \vdash [x \mapsto M{:}A]P \Leftarrow \text{prop}\ \big[[x \mapsto M']^{\text{p}}_A(P')\big]$.

6. If $\Delta; P \vdash E \Leftarrow x{:}A'.Q\ [E']$ and $\Delta, x{:}A'; Q \vdash F \Leftarrow y{:}B.R\ [F']$ where $x \notin \text{FV}(B) \cup \text{FV}(R)$,

   then $\Delta; P \vdash \langle x{:}A \mapsto E\rangle F \Leftarrow y{:}B.R\ [\langle x \mapsto E'\rangle_A(F')]$.

Proof: By simultaneous induction on the structure of the derivations.

Theorem 15: Basic mathematical properties

The following sequents can be derived by the rules in the assertion logic:

1. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{nat}}(M_1 + N, M_2 + N) \vdash \text{id}_{\text{nat}}(M_1, M_2), \Gamma_2$
2. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{nat}}(M + N, N) \vdash \text{id}_{\text{nat}}(M, \text{zero}), \Gamma_2$
3. $\Delta; \Psi; \Gamma_1 \vdash \text{id}_{\text{nat}}(M + N, N + M), \Gamma_2$
4. $\Delta; \Psi; \Gamma_1 \vdash \text{id}_{\text{nat}}(M \times N, N \times M), \Gamma_2$
5. $\Delta; \Psi; \Gamma_1 \vdash \text{id}_{\text{nat}}\big((M_1 + M_2) + M_3, M_1 + (M_2 + M_3)\big), \Gamma_2$
6. $\Delta; \Psi; \Gamma_1 \vdash \text{id}_{\text{nat}}\big((M_1 \times M_2) \times M_3, M_1 \times (M_2 \times M_3)\big), \Gamma_2$
7. $\Delta; \Psi; \Gamma_1 \vdash \text{id}_{\text{nat}}\big((M_1 + M_2) \times N, M_1 \times N + M_2 \times N\big), \Gamma_2$
8. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{nat}}(M + N, \text{zero}) \vdash \text{id}_{\text{nat}}(M, \text{zero}), \Gamma_2$
9. $\Delta; \Psi; \Gamma_1, M \leq N, N \leq M \vdash \text{id}_{\text{nat}}(M, N), \Gamma_2$
10. $\Delta; \Psi; \Gamma_1 \vdash M \leq N, N \leq M, \Gamma_2$
11. $\Delta; \Psi; \Gamma_1, M \leq N \vdash \text{id}_{\text{nat}}(M, N), M < N, \Gamma_2$
12. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{nat}}(M, \text{succ}\ M) \vdash \Gamma_2$
13. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{nat}}\big(M, \text{succ}(M + N)\big) \vdash \Gamma_2$
14. $\Delta; \Psi; \Gamma_1, M \leq N, N < M \vdash \Gamma_2$
15. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{bool}}(\text{eq}(M, N), \text{true}) \vdash \text{id}_{\text{nat}}(M, N), \Gamma_2$
16. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{bool}}(\text{eq}(M, N), \text{false}), \text{id}_{\text{nat}}(M, N) \vdash \Gamma_2$
17. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{bool}}(\text{le}(M, N), \text{true}) \vdash M \leq N, \Gamma_2$
18. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{bool}}(\text{le}(M, N), \text{false}), M \leq N \vdash \Gamma_2$
19. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{bool}}(\text{lt}(M, N), \text{true}) \vdash M < N, \Gamma_2$
20. $\Delta; \Psi; \Gamma_1, \text{id}_{\text{bool}}(\text{lt}(M, N), \text{false}), M < N \vdash \Gamma_2$

Values	v, l	$::= () \mid \lambda x. M \mid \mathrm{dia}\ E \mid \mathrm{true} \mid \mathrm{false} \mid \mathrm{zero} \mid \mathrm{succ}\, v$
Heaps	χ	$::= \cdot \mid \chi, l \mapsto_A v$
Continuations	κ	$::= \cdot \mid x{:}A.E; \kappa$
Control expressions	ρ	$::= \kappa \rhd E$
Abstract machines	μ	$::= \chi, \kappa \rhd E$

Figure 2: Syntax for operational semantics

Proof: These sequents can be derived by using the rule for induction on natural numbers:

$$\frac{\Delta \vdash M \Leftarrow \mathrm{nat}\,[M] \quad \Delta; \Psi; \Gamma_1, P \vdash [x \mapsto \mathrm{succ}\, x]^{\mathrm{p}}_{\mathrm{nat}}(P), \Gamma_2}{\Delta; \Psi; \Gamma_1, [x \mapsto \mathrm{zero}]^{\mathrm{p}}_{\mathrm{nat}}(P) \vdash [x \mapsto M]^{\mathrm{p}}_{\mathrm{nat}}(P), \Gamma_2}$$

To derive the first above sequent, for instance, we first use the cut rule to change the sequent to $\Delta; \Psi; \Gamma_1 \vdash \mathrm{id}_{\mathrm{nat}}(M_1 + N, M_2 + N) \supset \mathrm{id}_{\mathrm{nat}}(M_1, M_2), \Gamma_2$. Then, after applying induction on N, we have to derive the following two sequents:

- $\Delta; \Psi; \Gamma_1 \vdash \mathrm{id}_{\mathrm{nat}}(M_1, M_2) \supset \mathrm{id}_{\mathrm{nat}}(M_1, M_2), \Gamma_2$
- $\Delta; \Psi; \Gamma_1, \mathrm{id}_{\mathrm{nat}}(M_1 + n, M_2 + n) \supset \mathrm{id}_{\mathrm{nat}}(M_1, M_2) \vdash$

  $\mathrm{id}_{\mathrm{nat}}\big(\mathrm{succ}(M_1 + n), \mathrm{succ}(M_2 + n)\big) \supset \mathrm{id}_{\mathrm{nat}}(M_1, M_2), \Gamma_2$

The former is the case when $N = \mathrm{zero}$, which is shown by the identity principles. The latter is the inductive step, which is also easily derived by the id rule and other basic rules.

The last six sequents need nested induction to derive them. For example, the sequent 15 is derived from $\forall n{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{bool}}(\mathrm{eq}(m, n), \mathrm{true}) \supset \mathrm{id}_{\mathrm{nat}}(m, n)$, by induction first on m and second on n.

The sequent 10 should require the most complicated proof. This is derived by induction on m in $\exists p{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{nat}}(m + p, N) \vee \exists q{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{nat}}(N + q, m)$. The case when $m = \mathrm{zero}$ should be obvious. In the induction step, we need to show the following two:

- $\exists p{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{nat}}(m + p, N) \vdash \exists r{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{nat}}(\mathrm{succ}(m + r), N) \vee \exists s{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{nat}}(N + s, \mathrm{succ}\, m)$
- $\exists q{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{nat}}(N + q, m) \vdash \exists r{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{nat}}(\mathrm{succ}(m + r), N) \vee \exists s{:}\,\mathrm{nat}.\,\mathrm{id}_{\mathrm{nat}}(N + s, \mathrm{succ}\, m)$

For the latter, we simply instantiate q and give $\mathrm{succ}\, q$ as a witness to s. For the former, we need case analysis on p. In the case when $p = \mathrm{zero}$, the witness $s = \mathrm{succ}\,\mathrm{zero}$ suffices for $\mathrm{id}_{\mathrm{nat}}(N + s, \mathrm{succ}\, m)$, and in the case when $p > zero$, $\mathrm{succ}\, r = p$ suffices $\mathrm{id}_{\mathrm{nat}}(\mathrm{succ}(m + r), N)$. We actually need induction on p to perform case analysis within the assertion logic.

The other above sequents can be derived by similar inductions, therefore the proof for them are omitted.

5 Operational Semantics

In this section, an operational semantics for HTT is defined, to establish soundness with respect to evaluation.

Firstly, additional syntax is defined for the operational semantics as in Figure 2. The definition is the same as in [14]. A heap is a mapping from locations to values. A continuation is a sequence of computations with one parameter each, where every computation receives a parameter value and passes the result of the computation to the next one. A control expression is a continuation with a computation, where the result of the computation is passed to the continuation. An abstract machine is a pair of a heap and a control expression, which represents a state during evaluation.

Control expressions are needed to keep the order of computations. The typing judgment for control expressions has the form of $\Delta; P \vdash \kappa \rhd E \Leftarrow x{:}A.Q$, which is similar to that for computations. The typing judgment for abstract machines has the form of $\vdash \chi, \kappa \rhd E \Leftarrow x{:}A.Q$ and this is equivalent to $\cdot; [\![\chi]\!] \vdash \kappa \rhd E \Leftarrow x{:}A.Q$ by definition, where $[\![\chi]\!]$ is an expression which converts a heap into a proposition that expresses the heap:

$$\begin{aligned} [\![\cdot]\!] &:= \top \\ [\![\chi, l \mapsto_A v]\!] &:= [\![\chi]\!] \wedge \mathrm{seleq}_A(\mathrm{mem}, l, v) \end{aligned}$$

Evaluation is defined using three types of judgments; one for elim terms $K \hookrightarrow_{\mathrm{k}} K'$, another for intro terms $M \hookrightarrow_{\mathrm{m}} M'$, and the other for abstract machines $\chi, \kappa \rhd E \hookrightarrow_{\mathrm{e}} \chi', \kappa' \rhd E'$. Each judgment denotes one step of evaluation.

Lemma 16: Replacement

1. If $\Delta; P \vdash \kappa \rhd E \Leftarrow x{:}A.Q$, then $\Delta; P \vdash E \Leftarrow y{:}B.R\ [E']$ for some y, B, R, E' and if $\Delta_1; P_1 \vdash \kappa_1 \rhd E_1 \Leftarrow y{:}B.R$ for some Δ_1 extending Δ, then $\Delta_1; P \vdash \kappa_1; \kappa \rhd E_1 \Leftarrow x{:}A.Q$.
2. If $\Delta; P \vdash y{:}B.F; \kappa \rhd E \Leftarrow x{:}A.Q$, then $\Delta; P \vdash \kappa \rhd \langle y{:}B \mapsto E \rangle F \Leftarrow x{:}A.Q$.

Proof: By straightforward induction on the structure of κ.

Theorem 17: Preservation

1. If $K_0 \hookrightarrow_{\mathrm{k}} K_1$ and $\vdash K_0 \Rightarrow A\ [N']$, then $\vdash K_1 \Rightarrow A\ [N']$.
2. If $M_0 \hookrightarrow_{\mathrm{m}} M_1$ and $\vdash M_0 \Leftarrow A\ [M']$, then $\vdash M_1 \Leftarrow A\ [M']$.
3. If $\mu_0 \hookrightarrow_{\mathrm{e}} \mu_1$ and $\vdash \mu_0 \Leftarrow x{:}\,A.\,Q$, then $\vdash \mu_1 \Leftarrow x{:}\,A.\,Q$.

Proof: The first two statements are proved by induction on the derivation of the evaluation judgment. The last one is proved by case analysis on the derivation of the typing judgment, using the first two statements and the replacement lemma. Here only the case when $\mu_0 = \chi_0, \kappa_0 \rhd y = \mathrm{alloc}\, v\, ; E$ is shown because the proof for the other cases are the same as in [14].

In this case, $\mu_1 = (\chi, l \mapsto_{\mathrm{unit}} (), \dots, l + v - 1 \mapsto_{\mathrm{unit}} ()), \kappa \rhd [x \mapsto l{:}\, \mathrm{nat}]E$. From the typing of μ_0, we have $\cdot; [\![\chi_0]\!] \vdash \kappa_0 \rhd y = \mathrm{alloc}\, v\, ; E \Leftarrow x{:}\,A.\,Q$. By the replacement lemma, there ist z, C, S, E' such that $\cdot; [\![\chi_0]\!] \vdash y = \mathrm{alloc}\, v\, ; E \Leftarrow z{:}\,C.\,S\ [E']$. By the typing rules for *alloc*, $y{:}\,\mathrm{nat}; [\![\chi_0]\!] \circ P \vdash E \Leftarrow z{:}\,C.\,S\ [E']$, where

$$P = \forall n{:}\,\mathrm{nat}.\big(n < l \vee l + v \leq n \supset \mathrm{share}(\mathrm{init}, \mathrm{mem}, n)\big) \wedge$$
$$\big(l \leq n \wedge n < l + v \supset \neg\mathrm{indom}(\mathrm{init}, n) \wedge \mathrm{seleq}_{\mathrm{unit}}(\mathrm{mem}, n, ())\big).$$

We can show $\cdot; \mathrm{mem}; [\![\chi_0, l \mapsto_{\mathrm{unit}} (), \dots, l + v - 1 \mapsto_{\mathrm{unit}} ()]\!] \vdash [\![\chi_0]\!] \circ P$ by a straightforward derivation and we get $y{:}\,\mathrm{nat}; [\![\chi_0, l \mapsto_{\mathrm{unit}} (), \dots, l + v - 1 \mapsto_{\mathrm{unit}} ()]\!] \circ P \vdash E \Leftarrow z{:}\,C.\,S\ [E']$ by properties of computations. From this $\vdash (\chi_0, l \mapsto_{\mathrm{unit}} (), \dots, l + v - 1 \mapsto_{\mathrm{unit}} ()), \kappa \rhd [x \mapsto l{:}\,\mathrm{nat}]E \Leftarrow z{:}\,C.\,S$ immediately follows. Q.E.D.

Definition 18: Heap soundness

For every heap χ and natural number l,

1. $\cdot; \mathrm{mem}; [\![\chi]\!] \vdash \mathrm{seleq}_A(\mathrm{mem}, l, -)$ implies $l \mapsto_A v \in \chi$ for some value v.
2. $\cdot; \mathrm{mem}; [\![\chi]\!] \vdash l \in \mathrm{mem}$ implies $l \mapsto_A v \in \chi$ for some type A and value v.

Heap soundness is an important property which is used to prove the progress theorem, but is not proved in this thesis. Although the statement of heap soundness may seem obvious, it cannot be proved simply by induction on χ because, in the case when $\chi = \cdot$, we need to prove that the sequent $\cdot; \mathrm{mem}; \top \vdash \mathrm{seleq}_A(\mathrm{mem}, l, -)$ can*not* be derived, which is hard by usual case analysis. Nanevski et al. [15] showed heap soundness for a HTT which was redefined

```
fillarray : Πp:nat. Πn:nat.
    { forall m:nat. m < n -> p + m in mem }
    dummy:unit
    { forall m:nat. (m < n -> seleq$_{\text{bool}}$(mem, p + m, true)) /\
                    (m < p \/ p + n <= m -> share(init, mem, m)) }
= λp. λn. dia(
    m' = loop$_{\text{nat}}^{\text{I}}$(zero,
        m.lt(m, n),
        m.({$P_0$} [p + m]$_{\text{bool}}$ = true; {$P_1$} succ m));
    ()
)
where I := forall i:nat.
    (i < n -> p + i in mem) /\
    (i < m' -> seleq$_{\text{bool}}$(mem, p + i, true)) /\
    (i < p \/ p + n <= i -> share(init, mem, i))
```

Figure 3: Example of array initialization

through a small-footprint approach, by constructing denotational semantics of HTT. Heap soundness was also proved by means of a set-theoretic interpretation of HTT [16]. Their proof may possibly be adapted for proof in this thesis, but it should require hard work and a lengthy description. Thus, heap soundness is left unproved in this thesis.

Theorem 19: Progress

1. If $\vdash K_0 \Rightarrow A\ [N']$, then either $K_0 = v{:}\,A$ or $K_0 \hookrightarrow_{\mathrm{k}} K_1$ for some K_1.
2. If $\vdash M_0 \Leftarrow A\ [M']$, then either $M_0 = v$ or $M_0 \hookrightarrow_{\mathrm{m}} M_1$ for some M_1.
3. If $\vdash \chi_0, \kappa_0 \rhd E_0 \Leftarrow x{:}\,A.\,Q$, then either $E_0 = v$ and $\kappa_0 = \cdot$ or $\chi_0, \kappa_0 \rhd E_0 \hookrightarrow_{\mathrm{e}} \chi_1, \kappa_1 \rhd E_1$ for some χ_1, κ_1, E_1.

Proof: The proof is done in the same manner in [14], by straightforward case analysis on K_0, M_0 and E_0. The proof relies on heap soundness, which is not proved in this thesis.

6 Examples

In this section, two programs are shown to illustrate how arrays can be used in programs of HTT. For brevity, the variable and heap contexts are omitted for most of the sequents that appear in this section.

6.1 Array initialization

The first example, Figure 3, is a simple function which fills the elements of an array with the

Boolean constant value true. The function takes two arguments, the first of which is the location of the first element and the second is the number of the elements. In the function is a loop, which is encapsulated in a monad. The result of the loop is unit because the function does not return a meaningful result.

The precondition of the function asserts that an array of length n starting at the location p is allocated when the computation starts. The postcondition asserts that the elements of the array are the Boolean value true and that locations other than the array are not changed during the computation. The function body is a simple loop, whose loop counter m starts with zero and goes up until $\mathrm{lt}(m, n)$ is not satisfied. In each iteration of the loop, the mth element is assigned true, and $\mathrm{succ}\, m$ is returned as the result of the iteration, which is the next loop counter value. The loop invariant I states a condition which is satisfied between iterations, that is, the array is allocated, elements with index up to the loop counter are assigned true, and other locations are not changed.

The propositions P_0 and P_1 that are yielded in typing are:

$P_0 = \mathrm{hid}(\mathrm{init}, \mathrm{mem}) \wedge [m' \mapsto m]I \wedge \mathrm{id}_{\mathrm{bool}}(\mathrm{lt}(m, n), \mathrm{true})$

$P_1 = P_0 \circ \mathrm{seleq}_{\mathrm{bool}}(\mathrm{mem}, p + m, \mathrm{true}) \wedge \forall n{:}\, \mathrm{nat}.\, \neg \mathrm{id}_{\mathrm{nat}}(n, p + m) \supset \mathrm{share}(\mathrm{init}, \mathrm{mem}, n).$

One of the sequents which must be proved during type checking is $P_1 \wedge \mathrm{id}_{\mathrm{nat}}(m', \mathrm{succ}\, m) \vdash I$, which confirms that the loop invariant is implied by the postcondition of the iterated computation. After applying some derivation rules, we find that this sequent can be derived from the following three sequents:

$$P_1, \mathrm{id}_{\mathrm{nat}}(m', \mathrm{succ}\, m), i < n \vdash p + i \in \mathrm{mem}$$

$$P_1, \mathrm{id}_{\mathrm{nat}}(m', \mathrm{succ}\, m), i < m' \vdash \mathrm{seleq}_{\mathrm{bool}}(\mathrm{mem}, p + i, \mathrm{true})$$

$$P_1, \mathrm{id}_{\mathrm{nat}}(m', \mathrm{succ}\, m), i < p \vee p + n \leq i \vdash \mathrm{share}(\mathrm{init}, \mathrm{mem}, \mathrm{i})$$

The first and third ones can be easily derived by basic rules of the assertion logic. The second one requires case analysis on i: the cases when $i < m$ and when $i = m$. Note that we do not have to consider the case when $i > m$ because $i < m' = \mathrm{succ}\, m$. We can replace $i < m'$ in the antecedent with $i \leq m$ by the cut and identity rules, and then replace it with $i < m \vee$

```
perm(h1, h2, p, n) :=
    forall i:nat.
        (i < p \/ p + n <= i -> share(h1, h2, i)) /\
        (i < n ->
            (exists j:nat. j < n /\ forall v:nat.
                seleq_nat(h1, p + i, v) -> seleq_nat(h2, p + j, v)) /\
            (exists j:nat. j < n /\ forall v:nat.
                seleq_nat(h2, p + i, v) -> seleq_nat(h1, p + j, v)))
largest(h, p, i) :=
    forall j:nat. j <= i ->
        exists v:nat. exists w:nat.
            seleq_nat(h, p + j, v) /\ seleq_nat(h, p + i, w) /\ v <= w
sorted(h, p, i, j) :=
    forall k:nat. i <= k /\ k < j -> largest(h, p, k)
```

Figure 4: Notations used in the sorting example

$\mathrm{id}_{\mathrm{nat}}(i, m)$ by the cut rule and the basic mathematical properties. Now we split the proposition into $i < m$ and $i = m$ then use basic derivation rules to show each of them.

6.2 Bubble sort

The second example is bubble sort. In this example, the function takes two arguments as the first example, but the array elements must be natural numbers. The function body is a doubly nested loop which implements bubble sort straightforwardly.

Here it is assumed that all the array elements are different in order to simplify the assertions. The more general case where multiple elements may be the same is discussed later.

In this example, the notations defined in Figure 4 are used as syntactic sugar to save space. $\mathrm{perm}(h_1, h_2, p, n)$ denotes that the only change between the two heaps h_1 and h_2 is permutation of the elements of the n-element array starting at p: that is, locations out of the array are not changed, and every element of the array in h_1 has a corresponding element in h_2 and vice versa. On the assumption that all the array elements are different, this establishes a one-to-one correspondence of the elements between the two heaps. $\mathrm{largest}(h, p, i)$ denotes that the ith element of the array starting at p in the heap h is not less than any preceding elements. $\mathrm{sorted}(h, p, i, j)$ denotes that the elements with index between i and j are sorted and not less than any preceding elements.

Using these notations, the type T of the sort function can be defined as:

```
bubblesort : T = λp. λn. dia(
    i' = loop_nat^I(zero,
        i. lt(i, n),
        i. (j' = loop_nat^J(zero,
                j. lt(i + succ j, n),
                j. (v = [p + j]_nat;
                    w = [p + succ j]_nat;
                    dummy' = if_unit(lt(w, v),
                        [p + j]_nat = w; [p + succ j]_nat = v; (),
                        ());
                    succ j));
            succ i));
    ()
)
where I :=
    perm(init, mem, p, n) /\
    (exists k:nat. id_nat(i' + k, n) /\ sorted(mem, p, k, n))
and J :=
    perm(init, mem, p, n) /\
    (exists k:nat. id_nat(i + k, n) /\ sorted(mem, p, k, n)) /\
    largest(mem, p, j') /\ i + succ j' <= n
```

Figure 5: Example of bubble sort

```
T := Πp:nat. Πn:nat.
    { forall i:nat. i < n -> seleq_nat(mem, p + i, -) }
    dummy:unit
    { perm(init, mem, p, n) /\ sorted(mem, p, zero, n) }
```

The precondition and the result type is the same as that of the first example, except that the array elements are restricted to natural numbers. The postcondition asserts that permutation of the given array is the only change in the heap during the computation and that the whole array is sorted (all the elements are arranged in ascending order). Note that the notations and the type above are independent of implementation of sort program and can be used for other sorting algorithms.

Now a straightforward implementation of bubble sort, which has the above type *T*, would be as in Figure 5. The structure of the program is simple, nested loops. The loop counter *i* of the outer loop starts at zero and goes up to the array length n. After the i^{th} iteration of the outer loop, the last *i* elements of the array are sorted (i.e. are not swapped any more), which is expressed as the proposition $\text{sorted}(\text{mem}, p, k, n)$ in the loop invariant *I*, where $k = n - i$. The loop counter *j* starts at zero and goes up to $n - i - 1$. In each iteration of the inner loop, the j^{th}

```
perm(h1, h2, p, n) :=
    exists pf:(Πn:nat.nat). biject(pf, n) /\
        forall i:nat.
            (i < p \/ p + n <= i -> share(h1, h2, i)) /\
            (i < n -> forall v:nat.
                seleq_nat(h1, p + i, v) <-> seleq_nat(h2, p + pf i, v))
biject(f, n) := forall k:nat. k < n ->
    (exists l:nat. l < n /\ id_nat(f l, k)) /\
    (forall l:nat. l < n -> id_nat(f k, f l) -> id_nat(k, l))
```

Figure 6: New notations for the type of sort function

and $(j+1)$th elements are looked up and compared. If the jth element is larger, the two elements are swapped. Then at the end of the iteration, it is shown that the $(j+1)$th element is not less than any preceding elements, which is expressed as $\mathrm{largest}(\mathrm{mem}, p, j')$ in J, as well as that it is not larger than any succeeding elements, expressed as a implication from $\mathrm{sorted}(\mathrm{mem}, p, k, n)$.

The proposition $i + \mathrm{succ}\, j' \leq n$ is required in J to show that the loop counter is exactly $n -$ ly $n - i - 1$ when the inner loop ends. Without this proposition, we can only show that the loop counter is not less than $n - i - 1$, but this is insufficient to show that the loop invariant of the outer loop is satisfied when an iteration of the outer loop ends. To show that the loop invariant holds, we need to show that the range of the array where sort is done is increased by one element during the inner loop. This is done by showing $\mathrm{largest}(\mathrm{mem}, p, j')$ and that j' is equal to $n - i'$.

In the above example, it is assumed that all the array elements are different. This is because it is hard to assert that there is no such case that more than one element before sort is corresponding to one element after sort. A possible solution to this is to use an additional array to track indices of the swapped elements. Let A be the array we want to sort, which may contain the same elements, and B be an additional array of the same length as that of A. Every element of B is initialized to a natural number equal to its index. When elements of A are swapped, elements of B are swapped correspondingly. Since the elements of B are all different, we can assert a one-to-one correspondence on them between before and after sort. By considering correspondence between the elements of A and of B, we can establish a one-to-one correspondence on the elements of A, which establishes that A is correctly sorted.

Another possible solution is to extend the type system and allow conditionals in terms. In the extended system, for example, a binary function that returns the minimal value of the arguments would be:

$$\lambda x.\lambda y.\,\mathrm{if}\ \mathrm{le}(x,y)\ \mathrm{then}\ x\ \mathrm{else}\ y : \Pi x{:}\,\mathrm{nat}.\,\Pi y{:}\,\mathrm{nat}.\,\mathrm{nat}$$

Now we can redefine $\mathrm{perm}(h_1, h_2, p, n)$ as in Figure 6. The correspondence of elements is established by showing existence of a bijective function which maps the index of an element before sort to that of after sort. The expression $\mathrm{biject}(f, n)$ means that the function f is bijective if we account the domain and the codomain of the function to be restricted to natural numbers less than n. Such a function can indeed be constructed as a term using conditionals. Since this function maps the indices of the elements (not elements themselves), the values of the elements are not important in verifying the assertion.

7 Related and Future Work

Nanevski et al., the pioneers of HTT, have proposed several variations of HTT to extend the capability of the type system for more sophisticated programs. In [15], polymorphism has been introduced and the notion of separation logic [17, 20] has been adapted to allow assertion about more complicated effects on heaps. In [16], HTT has been refined with the notion of Extended Calculus of Constructions [12], clearing away the syntactic distinction between terms, types and propositions. These improvements are mainly about fertility of types available in HTT. This thesis, on the other hand, improves HTT with respect to the capability of dealing with heaps.

Another direction of improvement in HTT may be as to the capability of terms. As illustrated in Section 6, extension of terms with operations, such as conditionals, enables more complicated calculation within terms. Since calculation available in the assertion logic depends on that of terms, improvement of the capability of terms will allow more complicated assertion.

Other recent researches on combination of functional and imperative programming include work by Honda et al. [10, 1], which integrates higher-order functions into Hoare logic, whereas

HTT integrates Hoare logic into functions. Krishnaswami [11] have proposed an extended version of separation logic, which is very similar to HTT but is so simple that it does not support pointer arithmetic. The type system by Birkedal et al. [2] also extends separation logic with higher-order procedures, though computation cannot return a value as the result in their system. Bulwahn et al. [3] have employed the monad of heap in their system and claimed applicability to practical verification tools, but it seems hard in their system to assert invariance of part of a heap that is not involved in a computation. Moreover, it seems impossible in these systems to allocate adjacent locations as an array.

8 Conclusions

In this thesis, an extended version of HTT was defined to allow typing of programs which deal with adjacent heap locations, where the number of updated locations may be specified at runtime. Changes from the original type system include redefinition of the *alloc* command, which allocates multiple adjacent locations, and addition of the *le* and *lt* operators, which are useful for loop condition. Treatment of heaps in the assertion logic was also redefined so that each predicate asserts about one location rather than the whole heap. As a result, adjacent multiple locations whose number is specified at runtime can be allocated and used as an array in HTT. This thesis also described how to assert inequality of natural numbers and how it is used in combination with quantifiers to assert about adjacent locations and their contents. Soundness of the extended HTT with respect to evaluation was established, assuming soundness of the assertion logic.

References

1. Berger, M. and Honda, K. and Yoshida, N.: A Logical Analysis of Aliasing in Imperative Higher-Order Functions, J. Func. Prog., Vol. 17, No. 4-5, pp. 473-546 (2007).
2. Birkedal, L. and Torp-Smith, N. and Yang, N.H.: Semantics of Separation-Logic Typing and Higher-Order Frame Rules, Proc. 20th LICS, pp. 260-269 (2005).
3. Bulwahn, L. and Krauss, A. and Haftmann, F. and Erkok, L. and Matthews, J.: Imperative Functional Programming with Isabelle/HOL, LNCS, Vol. 5170, pp, 134-149 (2008).
4. Cartwright, R. and Oppen, D.: The Logic of Aliasing, Acta Informatica, Vol. 15, No. 4, pp. 365-384 (1981).

5. Church, A.: A Formulation of the Simple Theory of Types, J. Symbolic Logic, Vol. 5, No. 1, pp. 56-68 (1940).
6. Coquand, T. and Huet, G.: The Calculus of Constructions, Inf. Comput., Vol. 76, No. 2-3, pp. 95-120 (1988).
7. Girard, J.Y.: Une extension de l'interpétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types, Proc. Scandinavian Logic Symposium, pp. 63-92 (1971).
8. Harper, R. and Honsell, F. and Plotkin, G.: A Framework for Defining Logics, J. ACM, Vol. 40, No. 1, pp. 143-184 (1993).
9. Hoare, C.A.R.: An Axiomatic Basis for Computer Programming, Comm. ACM, Vol. 12, No. 10, pp. 576-585 (1969).
10. Honda, K. and Yoshida, N. and Berger, M.: An Observationally Complete Program Logic for Imperative Higher-Order Functions, Proc. 20th LICS, pp. 270-279 (2005).
11. Krishnaswami, N.: Separation Logic for a Higher-Order Typed Language, Proc. 3rd SPACE, pp. 73-82 (2006).
12. Luo, Z.: An Extended Calculus of Constructions, PhD Thesis, University of Edinburgh (1990).
13. Moggi, E.: Notions of Computation and Monads, Inf. Comput., Vol. 93, No. 1, pp. 55-92 (1991).
14. Nanevski, A. and Morrisett, G.: Dependent Type Theory of Stateful Higher-Order Functions, Harvard Univ., TR-24-05 (2005).
15. Nanevski, A. and Morrisett, G. and Birkedal, L.: Polymorphism and Separation in Hoare Type Theory, Harvard University, TR-10-06 (2006).
16. Nanevski, A. and Ahmed, A. and Morrisett, G. and Birkedal, L.: Abstract Predicates and Mutable ADTs in Hoare Type Theory, Harvard Univ., TR-14-06 (2006).
17. O'Hearn, P. and Reynolds, J. and Yang, H.: Local Reasoning about Programs that Alter Data Structures, LNCS, Vol. 2142, pp. 1-19 (2001).
18. Pfenning, F. and Davies, R.: A Judgmental Reconstruction of Modal Logic, Mathematical Structures in Computer Science, Vol. 11, No. 4, pp. 511-540 (2001).
19. Reynolds, J.C.: Towards a Theory of Type Structure, Proc. Paris Colloquium on Programming, pp. 408-423 (1974).
20. Reynolds, J.C.: Separation Logic: A Logic for Shared Mutable Data Structures, Proc. 17th LICS, pp. 55-74 (2002).
21. Wadler, P.: Comprehending Monads, Proc. ACM Conf. on Lisp and Functional Programming, pp. 61-78 (1990).
22. Wadler, P. and Thiemann, P.: The marriage of Effects and Monads, ACM Trans. Comput. Logic, Vol. 4, No. 1, pp. 1-32 (2003).
23. Watkins, K. and Cervesato, I. and Pfenning, F. and Walker, D.: A Concurrent Logical Framework: The Propositional Fragment, LNCS, Vol. 3085, pp. 355-377 (2004).

Appendix 1. Type System of HTT

A 1.1. Hereditary substitution

$$
\begin{array}{lll}
[x \mapsto M]^{\mathrm{k}}_{S}(x) & := M :: S & \\
[x \mapsto M]^{\mathrm{k}}_{S}(y) & := y & \text{if } y \neq x \\
[x \mapsto M]^{\mathrm{k}}_{S}(K\,N) & := K'N' & \text{if } [x \mapsto M]^{\mathrm{k}}_{S}(K) = K' \text{ and } [x \mapsto M]^{\mathrm{k}}_{S}(N) = N' \\
[x \mapsto M]^{\mathrm{k}}_{S}(K\,N) & := O' :: S_2 & \text{if } [x \mapsto M]^{\mathrm{k}}_{S}(K) = \lambda y.\,M' :: S_1 \to S_2 \\
 & & \text{where } S_1 \to S_2 \preccurlyeq S \text{ and } [x \mapsto M]^{\mathrm{k}}_{S}(N) = N' \\
 & & \text{and } [y \mapsto N']^{\mathrm{k}}_{S_1}(M') = O' \\
[x \mapsto M]^{\mathrm{k}}_{A}(K) & := \text{fails} & \text{otherwise} \\
[x \mapsto M]^{\mathrm{m}}_{S}(K) & := K' & \text{if } [x \mapsto M]^{\mathrm{k}}_{S}(K) = K' \\
[x \mapsto M]^{\mathrm{m}}_{S}(K) & := N' & \text{if } [x \mapsto M]^{\mathrm{k}}_{S}(K) = N' :: S' \\
[x \mapsto M]^{\mathrm{m}}_{S}(()) & := () & \\
[x \mapsto M]^{\mathrm{m}}_{S}(\lambda y.\,N) & := \lambda y.\,N' & \text{if } [x \mapsto M]^{\mathrm{k}}_{S}(N) = N' \\
 & & \text{choosing } y \notin \mathrm{FV}(M) \cup \{x\} \\
[x \mapsto M]^{\mathrm{m}}_{S}(\text{dia } E) & := \text{dia } E' & \text{if } [x \mapsto M]^{\mathrm{e}}_{S}(E) = E' \\
[x \mapsto M]^{\mathrm{m}}_{S}(\text{true}) & := \text{true} & \\
[x \mapsto M]^{\mathrm{m}}_{S}(\text{false}) & := \text{false} & \\
[x \mapsto M]^{\mathrm{m}}_{S}(\text{zero}) & := \text{zero} & \\
[x \mapsto M]^{\mathrm{m}}_{S}(\text{succ } N) & := \text{succ } N' & \text{if } [x \mapsto M]^{\mathrm{m}}_{S}(N) = N' \\
[x \mapsto M]^{\mathrm{m}}_{S}(N_1 + N_2) & := \text{plus}(N_1', N_2') & \\
 & \multicolumn{2}{l}{\quad \text{if } [x \mapsto M]^{\mathrm{m}}_{S}(N_1) = N_1' \text{ and } [x \mapsto M]^{\mathrm{m}}_{S}(N_2) = N_2'} \\
[x \mapsto M]^{\mathrm{m}}_{S}(N_1 \times N_2) & := \text{times}(N_1', N_2') & \\
 & \multicolumn{2}{l}{\quad \text{if } [x \mapsto M]^{\mathrm{m}}_{S}(N_1) = N_1' \text{ and } [x \mapsto M]^{\mathrm{m}}_{S}(N_2) = N_2'} \\
[x \mapsto M]^{\mathrm{m}}_{S}\big(\text{eq}(N_1, N_2)\big) & := \text{equals}(N_1', N_2') & \\
 & \multicolumn{2}{l}{\quad \text{if } [x \mapsto M]^{\mathrm{m}}_{S}(N_1) = N_1' \text{ and } [x \mapsto M]^{\mathrm{m}}_{S}(N_2) = N_2'} \\
[x \mapsto M]^{\mathrm{m}}_{S}\big(\text{le}(N_1, N_2)\big) & := \text{lessequal}(N_1', N_2') & \\
 & \multicolumn{2}{l}{\quad \text{if } [x \mapsto M]^{\mathrm{m}}_{S}(N_1) = N_1' \text{ and } [x \mapsto M]^{\mathrm{m}}_{S}(N_2) = N_2'} \\
[x \mapsto M]^{\mathrm{m}}_{S}\big(\text{lt}(N_1, N_2)\big) & := \text{lessthan}(N_1', N_2') & \\
 & \multicolumn{2}{l}{\quad \text{if } [x \mapsto M]^{\mathrm{m}}_{S}(N_1) = N_1' \text{ and } [x \mapsto M]^{\mathrm{m}}_{S}(N_2) = N_2'} \\
[x \mapsto M]^{\mathrm{m}}_{S}(N) & := \text{fails} & \text{otherwise} \\
[x \mapsto M]^{\mathrm{e}}_{S}(N) & := N' & \text{if } [x \mapsto M]^{\mathrm{m}}_{S}(N) = N' \\
[x \mapsto M]^{\mathrm{e}}_{S}(\text{let dia } y = K \text{ in } E) & \multicolumn{2}{l}{:= \text{let dia } y = K' \text{ in } E'} \\
\multicolumn{3}{l}{\qquad\qquad \text{if } [x \mapsto M]^{\mathrm{k}}_{S}(K) = K' \text{ and } [x \mapsto M]^{\mathrm{e}}_{S}(E) = E' \text{ choosing } y \notin \mathrm{FV}(M) \cup \{x\}} \\
[x \mapsto M]^{\mathrm{e}}_{S}(\text{let dia } y = K \text{ in } E) & := F' & \\
\multicolumn{3}{l}{\qquad\qquad \text{if } [x \mapsto M]^{\mathrm{k}}_{S}(K) = \text{dia } F :: \Diamond S_1 \text{ and } [x \mapsto M]^{\mathrm{e}}_{S}(E) = E'} \\
\multicolumn{3}{l}{\qquad\qquad \text{and } \Diamond S_1 \preccurlyeq S \text{ and } \langle y \mapsto F \rangle_{S_1}(E')} \\
\multicolumn{3}{l}{\qquad\qquad \text{choosing } y \notin \mathrm{FV}(M) \cup \{x\}} \\
[x \mapsto M]^{\mathrm{e}}_{S}(E) & \text{fails} & \text{otherwise} \\
[x \mapsto M]^{\mathrm{a}}_{S}(\alpha) & := \alpha & \\
[x \mapsto M]^{\mathrm{a}}_{S}(\text{bool}) & := \text{bool} & \\
[x \mapsto M]^{\mathrm{a}}_{S}(\text{nat}) & := \text{nat} & \\
[x \mapsto M]^{\mathrm{a}}_{S}(\text{unit}) & := \text{unit} &
\end{array}
$$

$[x \mapsto M]^{\mathrm{a}}_{S}(\Pi y{:}\, A.B) \quad := \Pi y{:}\, A'.B'$

if $[x \mapsto M]^{\mathrm{a}}_{S}(A) = A'$ and $[x \mapsto M]^{\mathrm{a}}_{S}(B) = B'$ choosing $y \notin \mathrm{FV}(M) \cup \{x\}$

$[x \mapsto M]^{\mathrm{a}}_{S}(\{P\}y{:}\, A\{Q\}) \quad := \{P'\}y{:}\, A'\{Q'\}$

if $[x \mapsto M]^{\mathrm{a}}_{S}(A) = A'$ and $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$ and $[x \mapsto M]^{\mathrm{p}}_{S}(Q) = Q'$ choosing $y \notin \mathrm{FV}(M) \cup \{x\}$

$[x \mapsto M]^{\mathrm{p}}_{S}(\top) \quad := \top$

$[x \mapsto M]^{\mathrm{p}}_{S}(\bot) \quad := \bot$

$[x \mapsto M]^{\mathrm{p}}_{S}(P \wedge Q) \quad := P' \wedge Q' \quad$ if $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$ and $[x \mapsto M]^{\mathrm{p}}_{S}(Q) = Q'$

$[x \mapsto M]^{\mathrm{p}}_{S}(P \vee Q) \quad := P' \vee Q' \quad$ if $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$ and $[x \mapsto M]^{\mathrm{p}}_{S}(Q) = Q'$

$[x \mapsto M]^{\mathrm{p}}_{S}(P \supset Q) \quad := P' \supset Q' \quad$ if $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$ and $[x \mapsto M]^{\mathrm{p}}_{S}(Q) = Q'$

$[x \mapsto M]^{\mathrm{p}}_{S}(\neg P) \quad := \neg P' \quad$ if $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$

$[x \mapsto M]^{\mathrm{p}}_{S}\big(\mathrm{id}_{A}(N_1, N_2)\big) \quad := \mathrm{id}_{A'}(N_1', N_2')$

if $[x \mapsto M]^{\mathrm{a}}_{S}(A) = A'$ and $[x \mapsto M]^{\mathrm{m}}_{S}(N_1) = N_1'$ and $[x \mapsto M]^{\mathrm{m}}_{S}(N_2) = N_2'$

$[x \mapsto M]^{\mathrm{p}}_{S}\big(\mathrm{seleq}_{A}(h, N_1, N_2)\big) \quad := \mathrm{seleq}_{A'}(h, N_1', N_2')$

if $[x \mapsto M]^{\mathrm{a}}_{S}(A) = A'$ and $[x \mapsto M]^{\mathrm{m}}_{S}(N_1) = N_1'$ and $[x \mapsto M]^{\mathrm{m}}_{S}(N_2) = N_2'$

$[x \mapsto M]^{\mathrm{p}}_{S}(\forall \alpha{:}\, \mathrm{type}.P) \quad := \forall \alpha{:}\, \mathrm{type}.P' \quad$ if $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$

$[x \mapsto M]^{\mathrm{p}}_{S}(\exists \alpha{:}\, \mathrm{type}.P) \quad := \exists \alpha{:}\, \mathrm{type}.P' \quad$ if $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$

$[x \mapsto M]^{\mathrm{p}}_{S}(\forall h{:}\, \mathrm{heap}.P) \quad := \forall h{:}\, \mathrm{heap}.P' \quad$ if $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$

$[x \mapsto M]^{\mathrm{p}}_{S}(\exists h{:}\, \mathrm{heap}.P) \quad := \exists h{:}\, \mathrm{heap}.P' \quad$ if $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$

$[x \mapsto M]^{\mathrm{p}}_{S}(\forall y{:}\, A.P) \quad := \forall y{:}\, A'.P'$

if $[x \mapsto M]^{\mathrm{a}}_{S}(A) = A'$ and $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$ choosing $y \notin \mathrm{FV}(M) \cup \{x\}$

$[x \mapsto M]^{\mathrm{p}}_{S}(\exists y{:}\, A.P) \quad := \exists y{:}\, A'.P'$

if $[x \mapsto M]^{\mathrm{a}}_{S}(A) = A'$ and $[x \mapsto M]^{\mathrm{p}}_{S}(P) = P'$ choosing $y \notin \mathrm{FV}(M) \cup \{x\}$

$$\mathrm{plus}(M, N) := \begin{cases} N & \text{if } M = \mathrm{zero} \\ M & \text{if } N = \mathrm{zero} \\ \mathrm{succ}\big(\mathrm{plus}(M', N)\big) & \text{if } M = \mathrm{succ}\, M' \\ \mathrm{succ}\big(\mathrm{plus}(M, N')\big) & \text{if } N = \mathrm{succ}\, N' \\ M + N & \text{otherwise} \end{cases}$$

$$\mathrm{times}(M, N) := \begin{cases} \mathrm{zero} & \text{if } M = \mathrm{zero} \text{ or } N = \mathrm{zero} \\ \mathrm{plus}\big(M', \mathrm{times}(M', N)\big) & \text{if } M = \mathrm{succ}\, M' \\ \mathrm{plus}(\mathrm{times}(M, N'), N') & \text{if } N = \mathrm{succ}\, N' \\ M \times N & \text{otherwise} \end{cases}$$

$$\mathrm{equals}(M, N) := \begin{cases} \mathrm{true} & \text{if } M = N = \mathrm{zero} \\ \mathrm{false} & \text{if } M = \mathrm{zero} \text{ and } N = \mathrm{succ}\, N' \\ \mathrm{false} & \text{if } N = \mathrm{zero} \text{ and } M = \mathrm{succ}\, M' \\ \mathrm{equals}(M,' N') & \text{if } M' = \mathrm{succ}\, M' \text{ and } N = \mathrm{succ}\, N' \\ \mathrm{eq}(M, N) & \text{otherwise} \end{cases}$$

$$\mathrm{lessequal}(M, N) := \begin{cases} \mathrm{true} & \text{if } M = \mathrm{zero} \\ \mathrm{false} & \text{if } M = \mathrm{succ}\, M' \text{ and } N = \mathrm{zero} \\ \mathrm{lessequal}(M,' N') & \text{if } M' = \mathrm{succ}\, M' \text{ and } N = \mathrm{succ}\, N' \\ \mathrm{le}(M, N) & \text{otherwise} \end{cases}$$

$$\text{lessthan}(M, N) \coloneqq \begin{cases} \text{true} & \text{if } M = \text{zero and } N = \text{succ}\, N' \\ \text{false} & \text{if } N = \text{zero} \\ \text{lessthan}(M,' N') & \text{if } M' = \text{succ}\, M' \text{ and } N = \text{succ}\, N' \\ \text{lt}(M, N) & \text{otherwise} \end{cases}$$

$$\text{nat}^- \coloneqq \text{nat}, \text{bool}^- \coloneqq \text{bool}, \text{unit}^- \coloneqq \text{unit}$$
$$(\Pi x{:}\, A. B)^- \coloneqq A^- \to B^-$$
$$(\{P\}x{:}\, A\{Q\})^- \coloneqq \diamond(A^-)$$

A 1.2. Monadic substitution

$$\begin{array}{lll}
\langle x{:}\, A \mapsto M\rangle F & \coloneqq [x \mapsto M{:}\, A]F & \\
\langle x{:}\, A \mapsto \text{let dia}\, y = K \text{ in } E\rangle F & \coloneqq \text{let dia}\, y = K \text{ in } \langle x{:}\, A \mapsto E\rangle F & \\
\langle x{:}\, A \mapsto c; E\rangle F & \coloneqq c; \langle x{:}\, A \mapsto E\rangle F & \\
\langle x \mapsto M\rangle_S(F) & \coloneqq F' & \text{if } [x \mapsto M]^{\text{e}}_S(F) = F' \\
\langle x \mapsto \text{let dia}\, y = K \text{ in } E\rangle_S(F) & \coloneqq \text{let dia}\, y = K \text{ in } F' & \text{if } \langle x \mapsto E\rangle_S(F) = F' \\
\langle x \mapsto c; E\rangle_S(F) & \coloneqq c; F' & \text{if } \langle x \mapsto E\rangle_S(F) = F'
\end{array}$$

A 1.3. Context formation

$$\frac{}{\vdash \cdot \text{ ctx}} \quad \frac{\vdash \Delta \text{ ctx} \quad \Delta \vdash A \Leftarrow \text{type } [A]}{\vdash (\Delta, x{:}\, A) \text{ ctx}} \quad \frac{\vdash \Delta \text{ ctx}}{\vdash (\Delta, \alpha) \text{ ctx}}$$

$$\frac{}{\Delta; \Psi \vdash \cdot \text{ pctx}} \quad \frac{\Delta; \Psi \vdash \Gamma \text{ pctx} \quad \Delta; \Psi \vdash P \Leftarrow \text{prop } [P]}{\Delta; \Psi \vdash (\Gamma, P) \text{ pctx}}$$

A 1.4. Type formation

$$\frac{}{\Delta, \alpha, \Delta' \vdash \alpha \Leftarrow \text{type } [\alpha]} \quad \frac{}{\Delta \vdash \text{bool} \Leftarrow \text{type } [\text{bool}]}$$

$$\frac{}{\Delta \vdash \text{nat} \Leftarrow \text{type } [\text{nat}]} \quad \frac{}{\Delta \vdash \text{unit} \Leftarrow \text{type } [\text{unit}]}$$

$$\frac{\Delta \vdash A \Leftarrow \text{type } [A'] \quad \Delta, x{:}\, A \vdash B \Leftarrow \text{type } [B']}{\Delta \vdash \Pi x{:}\, A. B \Leftarrow \text{type } [\Pi x{:}\, A'. B']}$$

$$\frac{\Delta; \text{mem} \vdash P \Leftarrow \text{prop } [P'] \quad \Delta \vdash A \Leftarrow \text{type } [A'] \quad \Delta, x{:}\, A'; \text{init}, \text{mem} \vdash Q \Leftarrow \text{prop } [Q']}{\Delta \vdash \{P\}x{:}\, A\{Q\} \Leftarrow \text{type } [\{P'\}x{:}\, A'\{Q'\}]}$$

A 1.5. Proposition formation

$$\frac{\Delta \vdash A \Leftarrow \text{type } [A'] \quad \Delta \vdash M \Leftarrow A'[M'] \quad \Delta \vdash N \Leftarrow A'[N']}{\Delta; \Psi \vdash \text{id}_A(M, N) \Leftarrow \text{prop } [\text{id}_{A'}(M', N')]}$$

$$\frac{\begin{array}{cc} h \in \Psi & \Delta \vdash M \Leftarrow \text{nat } [M'] \\ \Delta \vdash A \Leftarrow \text{type } [A'] & \Delta \vdash N \Leftarrow A'[N'] \end{array}}{\Delta; \Psi \vdash \text{seleq}_A(h, M, N) \Leftarrow \text{prop } [\text{seleq}_{A'}(h, M', N')]}$$

$$\frac{}{\Delta; \Psi \vdash \top \Leftarrow \text{prop } [\top]} \quad \frac{}{\Delta; \Psi \vdash \bot \Leftarrow \text{prop } [\bot]}$$

$$\frac{\Delta; \Psi \vdash P \Leftarrow \text{prop } [P'] \quad \Delta; \Psi \vdash Q \Leftarrow \text{prop } [Q']}{\Delta; \Psi \vdash P \wedge Q \Leftarrow \text{prop } [P' \wedge Q']}$$

$$\frac{\Delta; \Psi \vdash P \Leftarrow \text{prop } [P'] \quad \Delta; \Psi \vdash Q \Leftarrow \text{prop } [Q']}{\Delta; \Psi \vdash P \vee Q \Leftarrow \text{prop } [P' \vee Q']}$$

$$\frac{\Delta; \Psi \vdash P \Leftarrow \text{prop } [P'] \quad \Delta; \Psi \vdash Q \Leftarrow \text{prop } [Q']}{\Delta; \Psi \vdash P \supset Q \Leftarrow \text{prop } [P' \supset Q']}$$

$$\frac{\Delta;\Psi \vdash P \Leftarrow \text{prop}\ [P']}{\Delta;\Psi \vdash \neg P \Leftarrow \text{prop}\ [\neg P']}$$

$$\frac{\Delta,\alpha;\Psi \vdash P \Leftarrow \text{prop}\ [P']}{\Delta;\Psi \vdash \forall\alpha{:}\,\text{type}.\,P \Leftarrow \text{prop}\ [\forall\alpha{:}\,\text{type}.\,P']} \quad \frac{\Delta,\alpha;\Psi \vdash P \Leftarrow \text{prop}\ [P']}{\Delta;\Psi \vdash \exists\alpha{:}\,\text{type}.\,P \Leftarrow \text{prop}\ [\exists\alpha{:}\,\text{type}.\,P']}$$

$$\frac{\Delta;\Psi,h \vdash P \Leftarrow \text{prop}\ [P']}{\Delta;\Psi \vdash \forall h{:}\,\text{heap}.\,P \Leftarrow \text{prop}\ [\forall h.\,\text{heap}.\,P']} \quad \frac{\Delta;\Psi,h \vdash P \Leftarrow \text{prop}\ [P']}{\Delta;\Psi \vdash \exists h{:}\,\text{heap}.\,P \Leftarrow \text{prop}\ [\exists h.\,\text{heap}.\,P']}$$

$$\frac{\Delta \vdash A \Leftarrow \text{type}\ [A'] \quad \Delta,x{:}\,A' \vdash P \Leftarrow \text{prop}\ [P']}{\Delta;\Psi \vdash \forall x{:}\,A.\,P \Leftarrow \text{prop}\ [\forall x{:}\,A'.\,P']} \quad \frac{\Delta \vdash A \Leftarrow \text{type}\ [A'] \quad \Delta,x{:}\,A' \vdash P \Leftarrow \text{prop}\ [P']}{\Delta;\Psi \vdash \exists x{:}\,A.\,P \Leftarrow \text{prop}\ [\exists x{:}\,A'.\,P']}$$

A 1.6. Typing of terms

$$\frac{}{\Delta \vdash () \Leftarrow \text{unıt}\ [()]} \quad \frac{}{\Delta \vdash \text{true} \Leftarrow \text{bool}\ [\text{true}]} \quad \frac{}{\Delta \vdash \text{false} \Leftarrow \text{bool}\ [\text{false}]}$$

$$\frac{}{\Delta \vdash \text{zero} \Leftarrow \text{nat}\ [\text{zero}]} \quad \frac{\Delta \vdash M \Leftarrow \text{nat}\ [M']}{\Delta \vdash \text{succ}\ M \Leftarrow \text{nat}\ [\text{succ}\ M']}$$

$$\frac{\Delta \vdash M \Leftarrow \text{nat}\ [M'] \quad \Delta \vdash N \Leftarrow \text{nat}\ [N']}{\Delta \vdash M + N \Leftarrow \text{nat}\ [\text{plus}(M',N')]} \quad \frac{\Delta \vdash M \Leftarrow \text{nat}\ [M'] \quad \Delta \vdash N \Leftarrow \text{nat}\ [N']}{\Delta \vdash M \times N \Leftarrow \text{nat}\ [\text{times}(M',N')]}$$

$$\frac{\Delta \vdash M \Leftarrow \text{nat}\ [M'] \quad \Delta \vdash N \Leftarrow \text{nat}\ [N']}{\Delta \vdash \text{eq}(M,N) \Leftarrow \text{bool}\ [\text{equals}(M',N')]} \quad \frac{\Delta \vdash M \Leftarrow \text{nat}\ [M'] \quad \Delta \vdash N \Leftarrow \text{nat}\ [N']}{\Delta \vdash \text{le}(M,N) \Leftarrow \text{bool}\ [\text{lessequal}(M',N')]}$$

$$\frac{\Delta \vdash M \Leftarrow \text{nat}\ [M'] \quad \Delta \vdash N \Leftarrow \text{nat}\ [N']}{\Delta \vdash \text{lt}(M,N) \Leftarrow \text{bool}\ [\text{lessthan}(M',N')]}$$

$$\frac{}{\Delta,x{:}\,A,\Delta' \vdash x \Rightarrow A\ [x]}\,\text{var}$$

$$\frac{\Delta,x{:}\,A \vdash M \Leftarrow B\ [M']}{\Delta \vdash \lambda x.\,M \Leftarrow \Pi x{:}\,A.\,B\ [\lambda x.\,M']}\,\Pi\text{I} \quad \frac{\Delta \vdash K \Rightarrow \Pi x{:}\,A.\,B\ [N'] \quad \Delta \vdash M \Leftarrow A\ [M']}{\Delta \vdash K\ M \Rightarrow [x \mapsto M']^{\text{a}}_{A}(B)\ [\text{apply}_A(N',M')]}\,\Pi\text{E}$$

$$\frac{\Delta \vdash K \Rightarrow A\ [N']}{\Delta \vdash K \Leftarrow A\ [\text{expand}_A(N')]} \Rightarrow\Leftarrow \quad \frac{\Delta \vdash A \Leftarrow \text{type}\ [A'] \quad \Delta \vdash M \Leftarrow A'\ [M']}{\Delta \vdash M{:}\,A \Rightarrow A'\ [M']} \Leftarrow\Rightarrow$$

$$\begin{array}{lll}
\text{apply}_A(K,M) & \coloneqq K\ M & \\
\text{apply}_A(\lambda x.\,N,M) & \coloneqq N' & \text{if } [x \mapsto M]^{\text{m}}_{A}(N) = N' \\
\text{apply}_A(N,M) & \text{fails} & \text{otherwise} \\
\text{expand}_{\text{bool}}(K) & \coloneqq K & \\
\text{expand}_{\text{nat}}(K) & \coloneqq K & \\
\text{expand}_{\text{unit}}(K) & \coloneqq K & \\
\text{expand}_{\Pi x:A.B}(K) & \coloneqq \lambda x.\,\text{expand}_B(K\ M) & \text{if expand}_A(x) = M \text{ choosing } x \notin \text{FV}(K) \\
\text{expand}_{\{P\}x:A\{Q\}}(K) & \coloneqq \text{dia}(\text{let dia } x = K \text{ in } M) & \text{if expand}_A(x) = M \\
\text{expand}_A(N) & \coloneqq N &
\end{array}$$

A 1.7. Typing of computations

$$\frac{\Delta;P \vdash E \Rightarrow x{:}\,A.\,R\ [E'] \quad \Delta,x{:}\,A;\text{init},\text{mem};R \vdash Q}{\Delta;P \vdash E \Leftarrow x{:}\,A.\,Q\ [E']}\,\text{consequent}$$

$$\frac{\Delta \vdash M \Leftarrow A\ [M']}{\Delta;P \vdash M \Rightarrow x{:}\,A.\,P \wedge \text{id}_A(\text{expand}_A(x),M')\ [M']}\,\text{comp}$$

$$\frac{\Delta; \mathrm{hid}(\mathrm{init}, \mathrm{mem}) \wedge P \vdash E \Leftarrow x{:}\, A.\, Q\ [E']}{\Delta \vdash \mathrm{dia}\ E \Leftarrow \{P\}x{:}\, A\{Q\}\ [\mathrm{dia}\ E']}\,\{\}\mathrm{I}$$

$$\frac{\Delta \vdash K \Rightarrow \{R_1\}x{:}\, A\{R_2\}\ [N'] \quad \Delta; \mathrm{init}, \mathrm{mem}; P \vdash R_1 \quad \Delta, x{:}\, A; P \circ R_2 \vdash E \Rightarrow y{:}\, B.\, Q\ [E']}{\Delta; P \vdash (\mathrm{let\ dia}\ x = K\ \mathrm{in}\ E) \Rightarrow y{:}\, B.\, (\exists x{:}\, A.\, Q)\ [\mathrm{reduce}_A(N', x, E')]}\,\{\}\mathrm{E}$$

$$\frac{\Delta \vdash M \Leftarrow \mathrm{nat}\ [M'] \quad \Delta, x{:}\, \mathrm{nat}; P \circ \mathrm{sp}(x = \mathrm{alloc}\ M') \vdash E \Rightarrow y{:}\, B.\, Q\ [E']}{\Delta; P \vdash (x = \mathrm{alloc}\ M; E) \Rightarrow y{:}\, B.\, (\exists x{:}\, \mathrm{nat}.\, Q)\ [x = \mathrm{alloc}\ M'; E']}\,\mathrm{alloc}$$

$$\frac{\begin{array}{c}\Delta \vdash A \Leftarrow \mathrm{type}\ [A'] \qquad \Delta; \mathrm{init}, \mathrm{mem}; P \vdash \mathrm{seleq}_{A'}(\mathrm{mem}, M', -) \\ \Delta \vdash M \Leftarrow \mathrm{nat}\ [M'] \quad \Delta, x{:}\, A; P \wedge \mathrm{sp}(x = [M']_{A'}) \vdash E \Rightarrow y{:}\, B.\, Q\ [E']\end{array}}{\Delta; P \vdash (x = [M]_A; E) \Rightarrow y{:}\, B.\, (\exists x{:}\, A'.\, Q)\ [x = [M']_{A'}; E']}\,\mathrm{lookup}$$

$$\frac{\begin{array}{l}\Delta \vdash A \Leftarrow \mathrm{type}\ [A'] \\ \Delta \vdash M \Leftarrow \mathrm{nat}\ [M'] \qquad \Delta; \mathrm{init}, \mathrm{mem}; P \vdash M' \in \mathrm{mem} \\ \Delta \vdash N \Leftarrow A'[M'] \quad \Delta; P \circ \mathrm{sp}([M']_{A'} = N') \vdash E \Rightarrow y{:}\, B.\, Q\ [E']\end{array}}{\Delta; P \vdash ([M]_A = N; E) \Rightarrow y{:}\, B.\, Q\ [[M']_{A'} = N'; E']}\,\mathrm{mutation}$$

$$\frac{\begin{array}{c}\Delta \vdash A \Leftarrow \mathrm{type}\ [A'] \quad \Delta; P \wedge \mathrm{id}_{\mathrm{bool}}(M', \mathrm{true}) \vdash E_1 \Rightarrow x{:}\, A'.\, P_1\ [E_1'] \\ \Delta \vdash M \Leftarrow \mathrm{bool}\ [M'] \quad \Delta; P \wedge \mathrm{id}_{\mathrm{bool}}(M', \mathrm{false}) \vdash E_2 \Rightarrow x{:}\, A'.\, P_2\ [E_2'] \\ \Delta, x{:}\, A'; P_1 \vee P_2 \vdash E \Rightarrow y{:}\, B.\, Q\ [E']\end{array}}{\Delta; P \vdash (x = \mathrm{if}_A(M, E_1, E_2); E) \Rightarrow y{:}\, B.\, (\exists x{:}\, A'.\, Q)\ [x = \mathrm{if}_{A'}(M', E_1', E_2'); E]}\,\mathrm{if}$$

$$\frac{\begin{array}{c}\Delta \vdash A \Leftarrow \mathrm{type}\ [A'] \\ \Delta \vdash M \Leftarrow A'\ [M'] \\ \Delta, x{:}\, A' \vdash N \Leftarrow \mathrm{bool}\ [N'] \\ \Delta, x{:}\, A', y{:}\, A'; \mathrm{init}, \mathrm{mem} \vdash I \Leftarrow \mathrm{prop}\ [I'] \\ \Delta; \mathrm{init}, \mathrm{mem}; P \vdash [x \mapsto M', y \mapsto M']^{\mathrm{p}}_{A'}([\mathrm{init} \mapsto \mathrm{mem}]I') \\ \Delta, x{:}\, A', y{:}\, A', z{:}\, A'; \mathrm{init}, \mathrm{mem}; [y \mapsto z]I' \circ [x \mapsto z]I' \vdash I' \\ \Delta, x{:}\, A', y{:}\, A'; \mathrm{init}, \mathrm{mem}; I' \vdash [\mathrm{init} \mapsto \mathrm{mem}, x \mapsto y]I' \\ \Delta, x{:}\, A'; \mathrm{hid}(\mathrm{init}, \mathrm{mem}) \wedge [y \mapsto x]I' \wedge \mathrm{id}_{\mathrm{bool}}(N', \mathrm{true}) \vdash F \Leftarrow y{:}\, A'.\, I'\ [F'] \\ \Delta, y{:}\, A'; P \circ [x \mapsto M']^{\mathrm{p}}_{A'}(I') \wedge \mathrm{id}_{\mathrm{bool}}([x \mapsto y]N', \mathrm{false}) \vdash E \Rightarrow z{:}\, C.\, Q\ [E']\end{array}}{\Delta; P \vdash (y = \mathrm{loop}^I_A(M, x.\, N, x.\, F); E) \Rightarrow z{:}\, C.\, (\exists y{:}\, A'.\, Q)\ \left[y = \mathrm{loop}^{I'}_{A'}(M', x.\, N', x.\, F'); E'\right]}\,\mathrm{loop}$$

$$\frac{\begin{array}{c}\Delta \vdash \Pi x{:}\, A.\, \{R_1\}y{:}\, B\{R_2\} \Leftarrow \mathrm{type}\ [\Pi x{:}\, A'.\, \{R_1'\}y{:}\, B'\{R_2'\}] \\ \Delta \vdash M \Leftarrow A'\ [M'] \\ \Delta; \mathrm{init}, \mathrm{mem}; P \vdash [x \mapsto M']^{\mathrm{p}}_{A'}(R_1') \\ \Delta, f{:}\, (\Pi x{:}\, A'.\, \{R_1'\}y{:}\, B'\{R_2'\}), x{:}\, A'; \mathrm{hid}(\mathrm{init}, \mathrm{mem}) \wedge R_1' \Leftarrow y{:}\, B'.\, R_2'\ [E'] \\ \Delta, y{:}\, [x \mapsto M']^{\mathrm{a}}_{A'}(B'); P \circ [x \mapsto M']^{\mathrm{p}}_{A'}(R_2') \vdash F \Rightarrow z{:}\, C.\, Q\ [F']\end{array}}{\Delta; P \vdash \left(y = \mathrm{fix}_{\Pi x:A.\{R_1\}y:B\{R_2\}}(f.\, x.\, E, M); F\right) \Rightarrow z{:}\, C.\, \left(\exists y{:}\, [x \mapsto M]^{\mathrm{p}}_{A'}(B').\, Q\right)\ [G]}\,\mathrm{fix}$$

where G is $y = \mathrm{fix}_{\Pi x:A'.\{R_1'\}y:B'\{R_2'\}}(f.\, x.\, E', M'); F'$

$$\begin{array}{lll}
P \circ Q & \coloneqq \exists h{:}\, \mathrm{heap}.\, [\mathrm{mem} \mapsto h]P \wedge [\mathrm{init} \mapsto h]Q & \\
\mathrm{sp}(x = \mathrm{alloc}\ M) & \coloneqq \forall n{:}\, \mathrm{nat}.\, \big(n < x \vee x + M \leq n \supset \mathrm{share}(\mathrm{init}, \mathrm{mem}, n)\big) \wedge & \\
 & \quad \big(x \leq n \wedge n < x + M \supset \neg\mathrm{indom}(\mathrm{init}, n) \wedge \mathrm{seleq}_{\mathrm{unit}}(\mathrm{mem}, n, ())\big) & \\
\mathrm{sp}(x = [M]_A) & \coloneqq \mathrm{seleq}_A\big(\mathrm{mem}, M, \mathrm{expand}_A(x)\big) & \\
\mathrm{sp}([M]_A = N) & \coloneqq \mathrm{seleq}_A(\mathrm{mem}, M, N) \wedge \forall n{:}\, \mathrm{nat}.\, \neg\mathrm{id}_{\mathrm{nat}}(n, M) \supset \mathrm{share}(\mathrm{init}, \mathrm{mem}, n) & \\
\mathrm{reduce}_A(K, x, E) & \coloneqq \mathrm{let\ dia}\ x = K\ \mathrm{in}\ E & \\
\mathrm{reduce}_A(\mathrm{dia}\ F, x, E) & \coloneqq E' & \mathrm{if}\ \langle x \mapsto F\rangle_A(E) = E'
\end{array}$$

$\mathrm{reduce}_A(N, x, E)$ fails otherwise

A 1.8. Sequent calculus

$$\frac{}{\Delta;\Psi;\Gamma_1, p \vdash p, \Gamma_2}\,\mathrm{init} \qquad \frac{\Delta;\Psi;\Gamma_1 \vdash P, \Gamma_2 \quad \Delta;\Psi;\Gamma_1, P \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \Gamma_2}\,\mathrm{cut}$$

$$\frac{\Delta;\Psi;\Gamma_1 \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, P \vdash \Gamma_2} \quad \frac{\Delta;\Psi;\Gamma_1 \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash P, \Gamma_2} \quad \frac{\Delta;\Psi;\Gamma_1, P, P \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, P \vdash \Gamma_2} \quad \frac{\Delta;\Psi;\Gamma_1 \vdash P, P, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash P, \Gamma_2}$$

$$\frac{}{\Delta;\Psi;\Gamma_1, \bot \vdash \Gamma_2} \quad \frac{}{\Delta;\Psi;\Gamma_1 \vdash \top, \Gamma_2}$$

$$\frac{\Delta;\Psi;\Gamma_1, P, Q \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, P \wedge Q \vdash \Gamma_2} \quad \frac{\Delta;\Psi;\Gamma_1 \vdash P, \Gamma_2 \quad \Delta;\Psi;\Gamma_1 \vdash Q, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash P \wedge Q, \Gamma_2}$$

$$\frac{\Delta;\Psi;\Gamma_1, P \vdash \Gamma_2 \quad \Delta;\Psi;\Gamma_1, Q \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, P \vee Q \vdash \Gamma_2} \quad \frac{\Delta;\Psi;\Gamma_1 \vdash P, Q, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash P \vee Q, \Gamma_2}$$

$$\frac{\Delta;\Psi;\Gamma_1 \vdash P, \Gamma_2 \quad \Delta;\Psi;\Gamma_1, Q \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, P \supset Q \vdash \Gamma_2} \quad \frac{\Delta;\Psi;\Gamma_1, P \vdash Q, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash P \supset Q, \Gamma_2}$$

$$\frac{\Delta;\Psi;\Gamma_1 \vdash P, \Gamma_2}{\Delta;\Psi;\Gamma_1, \neg P \vdash \Gamma_2} \quad \frac{\Delta;\Psi;\Gamma_1, P \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \neg P, \Gamma_2}$$

$$\frac{\Delta \vdash A \Leftarrow \mathrm{type}\,[A] \quad \Delta;\Psi;\Gamma_1, [\alpha \mapsto A]P \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, \forall\alpha{:}\,\mathrm{type}.\,P \vdash \Gamma_2} \quad \frac{\Delta, \alpha;\Psi;\Gamma_1 \vdash P, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \forall\alpha{:}\,\mathrm{type}.\,P, \Gamma_2}$$

$$\frac{\Delta, \alpha;\Psi;\Gamma_1, P \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, \exists\alpha{:}\,\mathrm{type}.\,P \vdash \Gamma_2} \quad \frac{\Delta \vdash A \Leftarrow \mathrm{type}\,[A] \quad \Delta;\Psi;\Gamma_1 \vdash [\alpha \mapsto A]P, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \exists\alpha{:}\,\mathrm{type}.\,P, \Gamma_2}$$

$$\frac{h' \in \Psi \quad \Delta;\Psi;\Gamma_1, [h \mapsto h']P \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, \forall h{:}\,\mathrm{heap}.\,P \vdash \Gamma_2} \quad \frac{\Delta;\Psi, h;\Gamma_1 \vdash P, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \forall h{:}\,\mathrm{heap}.\,P, \Gamma_2}$$

$$\frac{\Delta;\Psi, h;\Gamma_1, P \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, \exists h{:}\,\mathrm{heap}.\,P \vdash \Gamma_2} \quad \frac{h' \in \Psi \quad \Delta;\Psi;\Gamma_1 \vdash [h \mapsto h']P, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \exists h{:}\,\mathrm{heap}.\,P, \Gamma_2}$$

$$\frac{\Delta \vdash M \Leftarrow A\,[M] \quad \Delta;\Psi;\Gamma_1, [x \mapsto M]^{\mathrm{p}}_A(P) \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, \forall x{:}\,A.\,P \vdash \Gamma_2} \quad \frac{\Delta, x{:}\,A;\Psi;\Gamma_1 \vdash P, \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \forall x{:}\,A.\,P, \Gamma_2}$$

$$\frac{\Delta, x{:}\,A;\Psi;\Gamma_1, P \vdash \Gamma_2}{\Delta;\Psi;\Gamma_1, \exists x{:}\,A.\,P \vdash \Gamma_2} \quad \frac{\Delta \vdash M \Leftarrow A\,[M] \quad \Delta;\Psi;\Gamma_1 \vdash [x \mapsto M]^{\mathrm{p}}_A(P), \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \exists x{:}\,A.\,P, \Gamma_2}$$

$$\frac{}{\Delta;\Psi;\Gamma_1 \vdash \mathrm{id}_A(M, M), \Gamma_2} \quad \frac{\Delta;\Psi;\Gamma_1, \mathrm{id}_A(M, N) \vdash [x \mapsto N]^{\mathrm{p}}_A(p), \Gamma_2}{\Delta;\Psi;\Gamma_1, \mathrm{id}_A(M, N) \vdash [x \mapsto M]^{\mathrm{p}}_A(p), \Gamma_2}$$

$$\frac{\Delta, x{:}\,A;\Psi;\Gamma_1 \vdash \mathrm{id}_B(M, N), \Gamma_2}{\Delta;\Psi;\Gamma_1 \vdash \mathrm{id}_{\Pi x:A.B}(\lambda x.\,M, \lambda x.\,N), \Gamma_2}$$

$$\frac{}{\Delta;\Psi;\Gamma_1, \mathrm{seleq}_A(h, M, N_1), \mathrm{seleq}_A(h, M, N_2) \vdash \imath\mathrm{d}_A(N_1, N_2), \Gamma_2}$$

$$\frac{}{\Delta;\Psi;\Gamma_1, \imath\mathrm{d}_{\mathrm{nat}}(\mathrm{zero}, \mathrm{succ}\,M) \vdash \Gamma_2}$$

$$\frac{}{\Delta;\Psi;\Gamma_1, \imath\mathrm{d}_{\mathrm{nat}}(\mathrm{succ}\,M, \mathrm{succ}\,N) \vdash \imath\mathrm{d}_{\mathrm{nat}}(M, N), \Gamma_2}$$

$$\frac{\Delta \vdash M \Leftarrow \mathrm{nat}\,[M] \quad \Delta;\Psi;\Gamma_1, P \vdash [x \mapsto \mathrm{succ}\,x]^{\mathrm{p}}_{\mathrm{nat}}(P), \Gamma_2}{\Delta;\Psi;\Gamma_1, [x \mapsto \mathrm{zero}]^{\mathrm{p}}_{\mathrm{nat}}(P) \vdash [x \mapsto M]^{\mathrm{p}}_{\mathrm{nat}}(P), \Gamma_2}$$

$$\frac{}{\Delta;\Psi;\Gamma_1, \imath\mathrm{d}_{\mathrm{bool}}(\mathrm{true}, \mathrm{false}) \vdash \Gamma_2}$$

$$\frac{\Delta \vdash M \Leftarrow \mathrm{bool}\;[M]}{\Delta; \Psi; \Gamma_1, [x \mapsto \mathrm{true}]^{\mathrm{p}}_{\mathrm{bool}}(P), [x \mapsto \mathrm{false}]^{\mathrm{p}}_{\mathrm{bool}}(P) \vdash [x \mapsto M]^{\mathrm{p}}_{\mathrm{bool}}(P), \Gamma_2}$$

Appendix 2. Operational Semantics of HTT

A 2.1. Heap formation

$$\frac{}{\vdash \cdot \Leftarrow \mathrm{heap}} \qquad \frac{\vdash \chi \Leftarrow \mathrm{heap} \quad \vdash A \Leftarrow \mathrm{type}\;[A] \quad \vdash l \Leftarrow \mathrm{nat}\;[l] \quad \vdash v \Leftarrow A\;[M] \quad v \notin \mathrm{dom}(\chi)}{\vdash (\chi, l \mapsto_A v) \Leftarrow \mathrm{heap}}$$

A 2.2. Typing of control expressions

$$\frac{\Delta; P \vdash E \Leftarrow x{:}A.Q\;[E']}{\Delta; P \vdash \cdot \rhd E \Leftarrow x{:}A.Q} \qquad \frac{\begin{array}{cc}\Delta \vdash B \Leftarrow \mathrm{type}\;[B'] & \Delta; P \vdash \kappa \rhd E \Leftarrow y{:}B'.R \\ y \notin \mathrm{FV}(A) \cup \mathrm{FV}(Q) & \Delta, y{:}B'; R \vdash F \Leftarrow x{:}A.Q\;[F']\end{array}}{\Delta; P \vdash \kappa; (y{:}B.F; \cdot) \rhd E \Leftarrow x{:}A.Q}$$

A 2.3. Typing of abstract machines

$$\frac{\cdot; [\![\chi]\!] \vdash \kappa \rhd E \Leftarrow x{:}A.Q}{\vdash \chi, \kappa \rhd E \Leftarrow x{:}A.Q}$$

A 2.4. Evaluation

$$\frac{K \hookrightarrow_{\mathrm{k}} K'}{K\;M \hookrightarrow_{\mathrm{k}} K'M} \quad \frac{M \hookrightarrow_{\mathrm{m}} M'}{(v{:}A)M \hookrightarrow_{\mathrm{k}} (v{:}A)M'} \quad \frac{M \hookrightarrow_{\mathrm{m}} M'}{M{:}A \hookrightarrow_{\mathrm{k}} M'{:}A}$$

$$\frac{}{(\lambda x.M : \Pi x{:}A.B)\;v \hookrightarrow_{\mathrm{k}} [x \mapsto v{:}A]M : [x \mapsto v{:}A]B}$$

$$\frac{K \hookrightarrow_{\mathrm{k}} K'M}{K \hookrightarrow_{\mathrm{m}} K'M} \quad \frac{K \hookrightarrow_{\mathrm{k}} v{:}A}{K \hookrightarrow_{\mathrm{m}} v}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\mathrm{succ}\;M \hookrightarrow_{\mathrm{m}} \mathrm{succ}\;M'}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{M + N \hookrightarrow_{\mathrm{m}} M' + N} \quad \frac{N \hookrightarrow_{\mathrm{m}} N'}{v + N \hookrightarrow_{\mathrm{m}} v + N'} \quad \frac{}{v_1 + v_2 \hookrightarrow_{\mathrm{m}} \mathrm{plus}(v_1, v_2)}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{M \times N \hookrightarrow_{\mathrm{m}} M' \times N} \quad \frac{N \hookrightarrow_{\mathrm{m}} N'}{v \times N \hookrightarrow_{\mathrm{m}} v \times N'} \quad \frac{}{v_1 + v_2 \hookrightarrow_{\mathrm{m}} \mathrm{times}(v_1, v_2)}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\mathrm{eq}(M, N) \hookrightarrow_{\mathrm{m}} \mathrm{eq}(M', N)} \quad \frac{N \hookrightarrow_{\mathrm{m}} N'}{\mathrm{eq}(v, N) \hookrightarrow_{\mathrm{m}} \mathrm{eq}(v, N')} \quad \frac{}{\mathrm{eq}(v_1, v_2) \hookrightarrow_{\mathrm{m}} \mathrm{equals}(v_1, v_2)}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\mathrm{le}(M, N) \hookrightarrow_{\mathrm{m}} \mathrm{le}(M', N)} \quad \frac{N \hookrightarrow_{\mathrm{m}} N'}{\mathrm{le}(v, N) \hookrightarrow_{\mathrm{m}} \mathrm{le}(v, N')} \quad \frac{}{\mathrm{le}(v_1, v_2) \hookrightarrow_{\mathrm{m}} \mathrm{lessequal}(v_1, v_2)}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\mathrm{lt}(M, N) \hookrightarrow_{\mathrm{m}} \mathrm{lt}(M', N)} \quad \frac{N \hookrightarrow_{\mathrm{m}} N'}{\mathrm{lt}(v, N) \hookrightarrow_{\mathrm{m}} \mathrm{lt}(v, N')} \quad \frac{}{\mathrm{lt}(v_1, v_2) \hookrightarrow_{\mathrm{m}} \mathrm{lessthan}(v_1, v_2)}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\chi, \kappa \rhd M \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd M'} \quad \frac{}{\chi, x{:}A.E; \kappa \rhd v \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd [x \mapsto v{:}A]E'}$$

$$\frac{K \hookrightarrow_{\mathrm{k}} K'}{\chi, \kappa \rhd \mathrm{let\;dia}\;x = K\;\mathrm{in}\;E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd \mathrm{let\;dia}\;x = K'\;\mathrm{in}\;E}$$

$$\frac{}{\chi, \kappa \rhd \mathrm{let\;d\imath a}\;x = (\mathrm{d\imath a}\;F){:}\{P\}x{:}A\{Q\}\;\mathrm{\imath n}\;E \hookrightarrow_{\mathrm{e}} \chi, (x{:}A.E; \kappa) \rhd F}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\chi, \kappa \rhd x = \mathrm{alloc}\;M; E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd x = \mathrm{alloc}\;M'; E}$$

$$\frac{\{i | l \le i < l+v\} \cap \mathrm{dom}(\chi) = \emptyset}{\chi, \kappa \rhd x = \mathrm{alloc}\, v; E \hookrightarrow_{\mathrm{e}} (\chi, l \mapsto_{\mathrm{unit}} (), \ldots, l+v-1 \mapsto_{\mathrm{unit}} ()), \kappa \rhd [x \mapsto l : \mathrm{nat}]E}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\chi, \kappa \rhd x = [M]_A; E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd x = [M']_A; E}$$

$$\frac{\vdash A \Leftarrow \mathrm{type}\, [A'] \quad l \mapsto_{A'} v \in \chi}{\chi, \kappa \rhd x = [l]_A; E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd [x \mapsto v : A]E}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\chi, \kappa \rhd [M]_A = N; E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd [M]_A = N; E} \quad \frac{N \hookrightarrow_{\mathrm{m}} N'}{\chi, \kappa \rhd [v]_A = N; E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd [v]_A = N'; E}$$

$$\frac{\vdash A \Leftarrow \mathrm{type}\, [A']}{(\chi_1, l \mapsto_B v', \chi_2), \kappa \rhd [l]_A = v; E \hookrightarrow_{\mathrm{e}} (\chi_1, l \mapsto_{A'} v, \chi_2), \kappa \rhd E}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\chi, \kappa \rhd x = \mathrm{if}_A(M, E_1, E_2); E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd x = \mathrm{if}_A(M', E_1, E_2); E}$$

$$\frac{}{\chi, \kappa \rhd x = \mathrm{if}_A(\mathrm{true}, E_1, E_2); E \hookrightarrow_{\mathrm{e}} \chi, x : A.E; \kappa \rhd E_1}$$

$$\frac{}{\chi, \kappa \rhd x = \mathrm{if}_A(\mathrm{false}, E_1, E_2); E \hookrightarrow_{\mathrm{e}} \chi, x : A.E; \kappa \rhd E_2}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\chi, \kappa \rhd x = \mathrm{loop}_A^I(M, x.N, x.F); E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd x = \mathrm{loop}_A^I(M, x.N, x.F); E}$$

$$\frac{}{\begin{array}{c}\chi, \kappa \rhd x = \mathrm{loop}_A^I(v, x.N, x.F); E \\ \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd x = \mathrm{if}_A([x \mapsto v : A]N, \langle z : A \mapsto [x \mapsto v : A]F\rangle(y = \mathrm{loop}_A^I(z, x.N, x.F); y), v); E\end{array}}$$

$$\frac{M \hookrightarrow_{\mathrm{m}} M'}{\chi, \kappa \rhd y = \mathrm{fix}_A(f.x.F, M); E \hookrightarrow_{\mathrm{e}} \chi, \kappa \rhd y = \mathrm{fix}_A(f.x.F, M'); E}$$

$$\frac{}{\chi, \kappa \rhd y = \mathrm{fix}_{\Pi x:A.\{R_1\}y:B\{R_2\}}(f.x.E, v); E \hookrightarrow_{\mathrm{e}} \chi, (y : [x \mapsto v : A]B.F; \kappa) \rhd [x \mapsto v : A, f \mapsto N]E}$$

where $N = \lambda z.\, \mathrm{dia}\big(y = \mathrm{fix}_{\Pi x:A.\{R_1\}y:B\{R_2\}}(f.x.E.z); y\big) : \Pi x : A.\{R_1\}y : B\{R_2\}$