

1 Language \mathcal{L}

In this section we define an imperative language \mathcal{L} with memory allocation and deallocation primitives, and for simplification we only use pointers as values.

The syntax of the language \mathcal{L} is as follows.

$$\begin{aligned}
x, y, z, \dots \text{ (variables)} &\in \mathbf{Var} \\
s \text{ (statements)} &::= \mathbf{skip} \mid s_1; s_2 \mid *x \leftarrow y \mid \mathbf{free}(x) \\
&\mid \mathbf{let } x = \mathbf{malloc}() \text{ in } s \mid \mathbf{let } x = \mathbf{null} \text{ in } s \\
&\mid \mathbf{let } x = y \text{ in } s \mid \mathbf{let } x = *y \text{ in } s \\
&\mid \mathbf{ifnull}(*x) \text{ then } s_1 \text{ else } s_2 \mid f(\vec{x}) \\
&\mid \mathbf{const}(*x)s \mid \mathbf{endconst}(*x) \\
d \text{ (proc. defs.)} &::= \{f \mapsto (x_1, \dots, x_n)s\} \\
D \text{ (definitions)} &::= \langle d_1 \cup \dots \cup d_n \rangle \\
P \text{ (programs)} &::= \langle D, s \rangle \\
E \text{ (context)} &::= E; s \mid []
\end{aligned}$$

Notation \vec{x} is for a finite sequence $\{x_1, \dots, x_n\}$, where we assume that each element is distinct; $[\vec{x}'/\vec{x}]s$ is for a term obtained by replacing each free occurrence of \vec{x} in s with variables \vec{x}' ; the $\mathbf{Dom}(f)$ is a mapping from function name f to its domain; for a map f , the $f\{x \mapsto v\}$ and $f \setminus x$ are defined as follows:

$$\begin{aligned}
f\{x \mapsto v\}(w) &= \begin{cases} v & \text{if } x = w \\ f(w) & \text{otherwise.} \end{cases} \\
(f \setminus x)(w) &= \begin{cases} \text{undefined} & \text{if } x = w \\ f(w) & \text{otherwise.} \end{cases}
\end{aligned}$$

and $\mathit{filter_C}(C, *x)$ is defined by a pseudocode as follows:

$$\begin{aligned}
\mathit{filter_C}(C, *x) &= \text{let } C' = C - \mathbf{const}(*x) \text{ in} \\
&\text{if } \mathbf{const}(*x) \in C' \text{ then return } C' \\
&\text{else return } C' \setminus \{\mathbf{null}(*x), -\mathbf{null}(*x)\}
\end{aligned}$$

The \mathbf{Var} is a countably infinite set of *variables* and each variable is a pointer. The statement \mathbf{skip} means "does nothing". The statement $s_1; s_2$ is a sequential execution of s_1 and s_2 . The statement $*x \leftarrow y$ updates the content of cell which is pointed to by x with the value y . The statement $\mathbf{free}(x)$ deallocates a memory cell which is pointed to by pointer x . The statement $\mathbf{let } x = e \text{ in } s$ evaluates the expression e , binds x to the result, and executes s . The expression $\mathbf{malloc}()$ allocates a new memory cell. The expression \mathbf{null} evaluates to the null pointer. The expression $*y$ means dereferencing a memory cell pointed to by y . The statement $\mathbf{ifnull}(*x) \text{ then } s_1 \text{ else } s_2$ executes s_1 if $*x$ is \mathbf{null} and executes s_2 otherwise. The statement $f(\vec{x})$ expresses a procedure f with arguments \vec{x} . The statement $\mathbf{const}(*x)s$ means $(*x)$ is a constant in statement s ; the statement $\mathbf{endconst}(*x)$ means from this point $(*x)$ maybe not constant.

The d represents a procedure definition which maps a procedure name f to its procedure body $(\vec{x})s$; The D represents a set of procedure definitions $\langle d_1 \cup \dots \cup d_n \rangle$, and each definition is distinct;

The pair $\langle D, s \rangle$ represents a program, where D is a set of definitions and s is a main statement; the E represents evaluation context.

1.1 Operational semantics

In this section we introduce operational semantics of language \mathcal{L} . We assume there is a countable infinite set \mathcal{H} of *heap addresses* ranged over by l .

We use a configuration $\langle H, R, s, n, C \rangle$ to express a run-time state. Each elements in the configuration is as follows.

- H , a *heap*, is a finite mapping from \mathcal{H} to $\mathcal{H} \cup \{\mathbf{null}\}$;
- R , an *environment*, is a finite mapping from **Var** to $\mathcal{H} \cup \{\mathbf{null}\}$;
- s is the statement that is being executed;
- n is a natural number that represents the number of memory cells available for allocation.
- C is a set of actions, which contains **const**($*x$), **null**($*x$) and $\neg\mathbf{null}(*x)$.

The operational semantics of the language \mathcal{L} is given by a labeled transition relation $\langle H, R, s, n, C \rangle \xrightarrow{\rho}_D \langle H', R', s', n', C' \rangle$. The label ρ is as follows.

$$\rho \text{ (label)} ::= \mathbf{malloc}(x') \mid \mathbf{free} \mid \tau$$

The ρ , an *action*, is **malloc**, **free**, or τ . The action **malloc** expresses an allocation of a memory cell; **free** expresses a deallocation of a memory cell; τ expresses the other actions. We often omit τ in $\xrightarrow{\tau}_D$. We use a metavariable σ for a finite sequence of actions $\rho_1 \dots \rho_n$. We write $\xrightarrow{\rho_1 \dots \rho_n}_D$ for $\xrightarrow{\rho_1}_D \xrightarrow{\rho_2}_D \dots \xrightarrow{\rho_n}_D$. We write $\xRightarrow{\rho}_D$ for $\xrightarrow{*}_D \xrightarrow{\rho}_D \xrightarrow{*}_D$. We write $\xRightarrow{\rho_1 \dots \rho_n}_D$ for $\xRightarrow{\rho_1}_D \dots \xRightarrow{\rho_n}_D$.

Figure 1 depicts the relation $\xrightarrow{\rho}_D$. Several important rules are listed as follows.

- SEM-CONSTSKIP: That a memory cell pointed to by x is no longer a constant is expressed by doing nothing.
- SEM-CONSTSEQ: That a memory cell pointed to by x should be a constant in a statement s is expressed by adding a statement **endconst**($*x$) at the end of statement s .
- SEM-FREE: Deallocation of a memory cell pointed to by x is expressed by deleting the entry for $R(x)$ from the heap. This action increments the number of available cells (i.e., n) by one (i.e., $n + 1$).
- SEM-MALLOC and SEM-OUTOFMEM: Allocation of a memory cell is expressed by adding a fresh entry to the heap. This action is allowed only if the number of available cells is positive; if the number is zero, then the configuration leads to an error state **OutOfMemory**.
- SEM-ASSIGNEXN, SEM-FREEEXN, SEM-DEREFEXN and SEM-FREEEXN : These rules express an illegal access to memory. If such action is performed, then the configuration leads to exceptional state **MemEx**. This state **MemEx** is not seen as an erroneous state in the current paper, hence a well-typed program may lead to these states. The command **free**(x) , if x is a null pointer, leads to **MemEx** in the current semantics, although it is equivalent to **skip** in the C language.

- SEM-CONSTEXN: expresses that if a constant $*x$ is changed in s it will raise **ConstEx** exception.

Our goal is to guarantee *total* memory-leak freedom and reject memory leaks. By our language \mathcal{L} , they are formally defined as follows:

Definition 1 (total memory-leak freedom). *A program $\langle D, s \rangle$ is totally memory-leak free if there is a natural number n such that it does not require more than n cells.*

Definition 2 (Memory leak). *A configuration $\langle H, R, s, n, C \rangle$ goes overflow if there is σ such that $\langle H, R, s, n, C \rangle \xrightarrow{\sigma} \text{OutOfMemory}$. A program $\langle D, s \rangle$ consumes at least n cells if $\langle \emptyset, \emptyset, s, n, \emptyset \rangle$ goes overflow.*

2 Type system

2.1 Types

The syntax of the types is as follows.

P (behavioral types)	$::=$	$\mathbf{0} \mid P_1; P_2 \mid \mathbf{free} \mid \alpha \mid \mu\alpha.P$ $\mid \mathbf{let } x = y \mathbf{ in } P \mid \mathbf{let } x = \mathbf{malloc} \mathbf{ in } P$ $\mid \mathbf{let } x = \mathbf{null} \mathbf{ in } P \mid \mathbf{let } x = *y \mathbf{ in } P$ $\mid (*x)(P_1, P_2) \mid \mathbf{const}(*x)P \mid \mathbf{endconst}(*x)$
Γ (variable type environment)	$::=$	$\{x_1, x_2, \dots, x_n\}$
Ψ (dependent function type)	$::=$	$(\vec{x})P$
Θ (function type environment)	$::=$	$\{f_1 : \Psi_1, \dots, f_n : \Psi_n\}$
k (constant values)	$::=$	$\mathbf{null}(*x) \mid \neg\mathbf{null}(*x) \mid \mathbf{const}(*x)$
F (constant value environment)	$::=$	$\{k_1, \dots, k_n\}$

Behavioral types ranged over by P express the abstraction of behaviors of a program. The type $\mathbf{0}$ represents the do-nothing behavior; the type $P_1; P_2$ represents the sequential execution of P_1 and P_2 ; The type **malloc** represents an allocation of a memory cell exactly once; the type **free** represents a deallocation; the type $\mu\alpha.P$ represents the behavior of α defined by the recursive equation $\alpha = P$; the type $(*x)(P_1, P_2)$ represents that P_1 or P_2 is obtained dependent on $*x$; the type $P_1 + P_2$ represents the choice between P_1 and P_2 ; the α is a type variable; the type $\mathbf{const}(*x)P$ represents that $*x$ is a constant in behavioral type P ; the type $\mathbf{endconst}(*x)$ represents $*x$ no longer be a constant from this point.

A type environments for variables ranged over by Γ is a set of variables. Since our interest is the behavior of a program, not the types of values, a variable type environment does not carry information on the types of variables.

Dependent function types ranged over by Ψ represents the behavior of a function; \vec{x} is the formal arguments of the function.

Function types ranged over by Θ is a mapping from function names to dependent function types.

k represents constant values, where $\mathbf{null}(*x)$ represents $(*x)$ is a null pointer; $\neg\mathbf{null}(*x)$ represents $(*x)$ is not a null pointer; $\mathbf{const}(*x)$ represents $(*x)$ is a constant.

Constant value environment ranged over by F is a set of constant variables.

$$\begin{array}{c}
\frac{C' = \text{filter_}C(C, *x)}{\langle H, R, \text{endconst}(*x), n, C \rangle \rightarrow_D \langle H, R, \text{skip}, n, C' \rangle} \quad (\text{SEM-CONSTSKIP}) \\
\langle H, R, \text{const}(*x)s, n, C \rangle \rightarrow_D \langle H, R, s; \text{endconst}(*x), n, C \cup \{\text{const}(*x)\} \rangle \quad (\text{SEM-CONSTSEQ}) \\
\langle H, R, \text{skip}; s, n, C \rangle \rightarrow_D \langle H, R, s, n, C \rangle \quad (\text{SEM-SKIP}) \\
\frac{\langle H, R, s_1, n, C \rangle \xrightarrow{\rho}_D \langle H', R', s'_1, n', C' \rangle}{\langle H, R, s_1; s_2, n, C \rangle \xrightarrow{\rho}_D \langle H', R', s'_1; s_2, n', C' \rangle} \quad (\text{SEM-SEQ}) \\
\frac{x' \notin \text{Dom}(R)}{\langle H, R, \text{let } x = \text{null in } s, n, C \rangle \rightarrow_D \langle H, R \{x' \mapsto \text{null}\}, [x'/x]s, n, C \rangle} \quad (\text{SEM-LETNULL}) \\
\frac{x' \notin \text{Dom}(R)}{\langle H, R, \text{let } x = y \text{ in } s, n, C \rangle \rightarrow_D \langle H, R \{x' \mapsto R(y)\}, [x'/x]s, n, C \rangle} \quad (\text{SEM-LETEQ}) \\
\frac{H(R(x)) = \text{null}, \text{const}(*x) \notin C}{\langle H, R, \text{ifnull}(*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle \rightarrow_D \langle H, R, s_1, n, C \rangle} \quad (\text{SEM-IFNULLT}) \\
\frac{H(R(x)) \neq \text{null}, \text{const}(*x) \notin C}{\langle H, R, \text{ifnull}(*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle \rightarrow_D \langle H, R, s_2, n, C \rangle} \quad (\text{SEM-IFNULLF}) \\
\frac{H(R(x)) = \text{null}, \text{const}(*x) \in C}{\langle H, R, \text{ifnull}(*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle \rightarrow_D \langle H, R, s_1, n, C \cup \{\text{null}(*x)\} \rangle} \quad (\text{SEM-IFCONSTNULLT}) \\
\frac{H(R(x)) \neq \text{null}, \text{const}(*x) \in C}{\langle H, R, \text{ifnull}(*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle \rightarrow_D \langle H, R, s_2, n, C \cup \{\neg \text{null}(*x)\} \rangle} \quad (\text{SEM-IFCONSTNULLF}) \\
\frac{\text{const}(*x) \notin C}{\langle H \{R(x) \mapsto v\}, R, *x \leftarrow y, n, C \rangle \rightarrow_D \langle H \{R(x) \mapsto R(y)\}, R, \text{skip}, n, C \rangle} \quad (\text{SEM-ASSIGN}) \\
\frac{x' \notin \text{Dom}(R) \quad R(y) \in \text{Dom}(H)}{\langle H, R, \text{let } x = *y \text{ in } s, n, C \rangle \rightarrow_D \langle H, R \{x' \mapsto H(R(y))\}, [x'/x]s, n, C \rangle} \quad (\text{SEM-LETDEREF}) \\
\frac{R(x) \neq \text{null} \text{ and } R(x) \in \text{Dom}(H)}{\langle H \{R(x) \mapsto v\}, R, \text{free}(x), n, C \rangle \xrightarrow{\text{free}}_D \langle H \setminus R(x), R, \text{skip}, n+1, C \rangle} \quad (\text{SEM-FREE}) \\
\frac{l \notin \text{Dom}(H) \quad n > 0}{\langle H, R, \text{let } x = \text{malloc}() \text{ in } s, n, C \rangle \xrightarrow{\text{malloc}(x')}_D \langle H \{l \mapsto v\}, R \{x' \mapsto l\}, [x'/x]s, n-1, C \rangle} \quad (\text{SEM-MALLOC}) \\
\frac{D(f) = (\vec{y})s}{\langle H, R, f(\vec{x}), n, C \rangle \rightarrow_D \langle H, R, [\vec{x}/\vec{y}]s, n, C \rangle} \quad (\text{SEM-CALL}) \\
\frac{R(x) = \text{null} \text{ or } R(x) \notin \text{Dom}(H)}{\langle H, R, \text{free}(x), n, C \rangle \xrightarrow{\text{free}}_D \text{MemEx}} \quad (\text{SEM-FREEEXN}) \\
\frac{R(x) = \text{null} \text{ or } R(x) \notin \text{Dom}(H)}{\langle H, R, *x \leftarrow y, n, C \rangle \rightarrow_D \text{MemEx}} \quad (\text{SEM-ASSIGNEXN}) \\
\frac{R(y) = \text{null} \text{ or } R(y) \notin \text{Dom}(H)}{\langle H, R, \text{let } x = *y \text{ in } s, n, C \rangle \rightarrow_D \text{MemEx}} \quad (\text{SEM-DEREFEXN}) \\
\frac{4}{\forall z. \text{const}(*z) \in C \text{ and } R(x) = R(z)} \quad (\text{SEM-ASSIGNCONSTEXN}) \\
\langle H \{R(x) \mapsto v\}, R, *x \leftarrow y, n, C \rangle \rightarrow_D \text{ConstEx} \\
\langle H, R, \text{let } x = \text{malloc}() \text{ in } s, 0, C \rangle \xrightarrow{\text{malloc}(x')}_D \text{OutOfMemory} \quad (\text{SEM-OUTOFMEM})
\end{array}$$

Figure 1: Operational semantics of \mathcal{L} .

Figure 2 depicts semantics of behavioral types with dependent types, and they are given by the labeled transition system. The relation $\langle P, F \rangle \xrightarrow{\rho} \langle P', F' \rangle$ means that P can make an action ρ , and P turns into P' after it makes action ρ ; F and F' record constant value environment before and after action ρ respectively.

Notation $filter.T(F, *x)$ is defined by a pseudocode as follows:

$$\begin{aligned} filter.T(F, *x) &= \text{let } F' = F - \mathbf{const}(*x) \text{ in} \\ &\quad \text{if } \mathbf{const}(*x) \notin F' \text{ then return } (F' \setminus \{\mathbf{null}(*x), \neg\mathbf{null}(*x)\}) \\ &\quad \text{else return } F' \end{aligned}$$

2.2 Typing rules

The type judgment for statements is of the form $\Theta; \Gamma \vdash s : P$, which represents that under the function type environment Θ and the variable type environment Γ , the abstracted behavioral type of statement s is P .

Before showing typing rules for statements in Figure 3, we need explain several important definitions. The first one is $OK_n(P, F)$, a predicate, where P represents the behavior of a program which consumes at most n memory cells.

Definition 3 ($\#_\rho(\sigma)$). $\#_\rho(\sigma)$ is the number of ρ in the sequence σ .

Definition 4. $OK_n(P, F)$ holds if, (1) $\forall P'$ and σ . if $\langle P, F \rangle \xrightarrow{\sigma} \langle P', F' \rangle$, then $\#_m(\sigma) - \#_f(\sigma) \leq n$ and (2) $OK(F)$

Definition 5. $OK(F)$ holds if F does not contain both $\mathbf{null}(*x)$ and $\neg\mathbf{null}(*x)$.

Definition 6 (Subtyping). $F \vdash P_1 \leq P_2$ is the largest relation such that, for any P'_1, F' and ρ , if $\langle P_1, F \rangle \xrightarrow{\rho} \langle P'_1, F' \rangle$, then there exists P'_2 such that $\langle P_2, F \rangle \xRightarrow{\rho} \langle P'_2, F' \rangle$ and $F' \vdash P'_1 \leq P'_2$. We write $P_1 \leq P_2$ if $F \vdash P_1 \leq P_2$ for any F .

2.3 Type soundness

Theorem 2.1. If $\vdash \langle D, s \rangle : n$ for some n , then $\langle D, s \rangle$ is totally memory-leak free.

The proof is based on the following lemmas: preservation and lack of immediate overflow.

Definition 7. we write $\Theta; \Gamma \vdash \langle H, R, s, n, C \rangle : \langle P, F \rangle$, if $\Theta; \Gamma \vdash s : P$ and $OK_n(P, F)$ with $C \approx F$.

Lemma 2.2 (Preservation). suppose that $\Theta; \Gamma \vdash \langle H, R, s, n, C \rangle : \langle P, F \rangle$. If $\langle H, R, s, n, C \rangle \xrightarrow{\rho} \langle H', R', s', n', C' \rangle$ then $\exists P', F'$ s.t. (1) $\Theta; \Gamma \vdash \langle H', R', s', n', C' \rangle : \langle P', F' \rangle$ and (2) $\langle P, F \rangle \xRightarrow{\rho} \langle P', F' \rangle$.

Lemma 2.3 (Lack of immediate overflow). If $\Theta; \Gamma \vdash \langle H, R, s, n, C \rangle : \langle P, F \rangle$, then $\langle H, R, s, n, C \rangle \not\xrightarrow{\mathbf{malloc}} \mathbf{OutOfMemory}$.

$$\begin{array}{c}
\langle \mathbf{0}; P, F \rangle \rightarrow \langle P, F \rangle \quad (\text{TR-SKIP}) \\
\langle \mathbf{free}, F \rangle \xrightarrow{\mathbf{free}} \langle \mathbf{0}, F \rangle \quad (\text{TR-FREE}) \quad \langle \mu\alpha.P, F \rangle \rightarrow \langle [\mu\alpha.P/\alpha]P, F \rangle \quad (\text{TR-REC}) \\
\langle P_1 + P_2, F \rangle \rightarrow \langle P_1, F \rangle \quad (\text{TR-CHOICE L}) \quad \langle P_1 + P_2, F \rangle \rightarrow \langle P_2, F \rangle \quad (\text{TR-CHOICE R}) \\
\frac{\langle P_1, F \rangle \xrightarrow{\rho} \langle P'_1, F' \rangle}{\langle P_1; P_2, F \rangle \xrightarrow{\rho} \langle P'_1; P_2, F' \rangle} \quad (\text{TR-SEQ}) \\
\langle \mathbf{let } x = \mathbf{malloc} \mathbf{ in } P, F \rangle \xrightarrow{\mathbf{malloc}(x')} \langle [x'/x]P, F \rangle \quad (\text{TR-LETMALLOC}) \\
\langle \mathbf{let } x = y \mathbf{ in } P, F \rangle \rightarrow \langle [x'/x]P, F \rangle \quad (\text{TR-LETXY}) \\
\langle \mathbf{let } x = *y \mathbf{ in } P, F \rangle \rightarrow \langle [x'/x]P, F \rangle \quad (\text{TR-LETX*Y}) \\
\langle \mathbf{let } x = \mathbf{null} \mathbf{ in } P, F \rangle \rightarrow \langle [x'/x]P, F \rangle \quad (\text{TR-LETX}) \\
\langle \mathbf{const}(*x)P, F \rangle \rightarrow \langle P; \mathbf{endconst}(*x), F \cup \{\mathbf{const}(*x)\} \rangle \quad (\text{TR-CONST}) \\
\frac{F' = \mathit{filter} T(F, *x)}{\langle \mathbf{endconst}(*x), F \rangle \rightarrow \langle \mathbf{0}, F' \rangle} \quad (\text{TR-ENDCONST}) \\
\frac{\mathbf{const}(*x) \notin F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \rangle} \quad (\text{TR-NOTCONST1}) \quad \frac{\mathbf{const}(*x) \notin F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \rangle} \quad (\text{TR-NOTCONST2}) \\
\frac{\mathbf{null}(*x) \in F \quad \mathbf{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \rangle} \quad (\text{TR-NULLIN}) \quad \frac{\neg \mathbf{null}(*x) \in F \quad \mathbf{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \rangle} \quad (\text{TR-NNULLIN}) \\
\frac{\mathbf{null}(*x), \neg \mathbf{null}(*x) \notin F \quad \mathbf{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \cup \mathbf{null}(*x) \rangle} \quad (\text{TR-NNULLNOTIN1}) \\
\frac{\mathbf{null}(*x), \neg \mathbf{null}(*x) \notin F \quad \mathbf{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \cup \neg \mathbf{null}(*x) \rangle} \quad (\text{TR-NNULLNOTIN2})
\end{array}$$

Figure 2: semantics of behavioral types with dependent types.

$$\begin{array}{c}
\Theta; \Gamma \vdash \mathbf{skip} : \mathbf{0} \quad (\text{T-SKIP}) \qquad \frac{\Theta; \Gamma \vdash s_1 : P_1 \quad \Theta; \Gamma \vdash s_2 : P_2}{\Theta; \Gamma \vdash s_1; s_2 : P_1; P_2} \quad (\text{T-SEQ}) \\
\Theta; \Gamma, x, y \vdash *x \leftarrow y : \mathbf{0} \quad (\text{T-ASSIGN}) \qquad \Theta; \Gamma, x \vdash \mathbf{free}(x) : \mathbf{free} \quad (\text{T-FREE}) \\
\frac{\Theta; \Gamma, x \vdash s : P}{\Theta; \Gamma \vdash \mathbf{let } x = \mathbf{malloc}() \mathbf{ in } s : \mathbf{let } x = \mathbf{malloc} \mathbf{ in } P} \quad (\text{T-MALLOC}) \qquad \frac{\Theta; \Gamma, x, y \vdash s : P}{\Theta; \Gamma, y \vdash \mathbf{let } x = y \mathbf{ in } s : \mathbf{let } x = y \mathbf{ in } P} \quad (\text{T-LETEQ}) \\
\frac{\Theta; \Gamma, x, y \vdash s : P}{\Theta; \Gamma, y \vdash \mathbf{let } x = *y \mathbf{ in } s : \mathbf{let } x = *y \mathbf{ in } P} \quad (\text{T-LETDEREF}) \qquad \frac{\Theta; \Gamma, x \vdash s : P}{\Theta; \Gamma \vdash \mathbf{let } x = \mathbf{null} \mathbf{ in } s : \mathbf{let } x = \mathbf{null} \mathbf{ in } P} \quad (\text{T-LETNULL}) \\
\Theta; \Gamma, x \vdash \mathbf{endconst}(*x) : \mathbf{endconst}(*x) \quad (\text{T-ENDCONST}) \\
\frac{\Theta; \Gamma, x \vdash s : P}{\Theta; \Gamma, x \vdash \mathbf{const}(*x)s : \mathbf{const}(*x)P} \quad (\text{T-CONST}) \\
\frac{\Theta; \Gamma, x \vdash s_1 : P_1 \quad \Theta; \Gamma, x \vdash s_2 : P_2}{\Theta; \Gamma, x \vdash \mathbf{ifnull}(*x) \mathbf{ then } s_1 \mathbf{ else } s_2 : (*x)(P_1, P_2)} \quad (\text{T-IFNULL}) \\
\Theta, f : (\vec{y})P; \Gamma, \vec{x} \vdash f(\vec{x}) : P[\vec{x}/\vec{y}] \quad (\text{T-CALL}) \\
\frac{\Theta; \Gamma \vdash s : P_1 \quad P_1 \leq P_2}{\Theta; \Gamma \vdash s : P_2} \quad (\text{T-SUB}) \\
\frac{\Theta(f) = (\vec{x})P \quad \mathbf{Dom}(D) = \mathbf{Dom}(\Theta) \quad \Theta; x_1, \dots, x_n \vdash s : P \text{ for each } f \mapsto (x_1, \dots, x_n)s \in D}{\vdash D : \Theta} \quad (\text{T-DEF}) \\
\frac{\vdash D : \Theta \quad \Theta; \emptyset \vdash s : P \quad OK_n(P, F)}{\vdash \langle D, s \rangle : n} \quad (\text{T-PROGRAM})
\end{array}$$

Figure 3: typing rules

3 Proof of Lemmas

Lemma 3.1. *If $\langle P, F \rangle \xrightarrow{\rho} \langle P', F' \rangle$ and $OK(F)$, then $OK(F')$*

Proof. By induction on $\langle P, F \rangle \xrightarrow{\rho} \langle P', F' \rangle$.

- Case $P = \mathbf{0}; P'$ and $\langle \mathbf{0}; P', F \rangle \rightarrow \langle P', F \rangle$
We need to prove $OK(F')$. From assumption, we have that $OK(F)$ holds, and in this case F' is the same as F . Therefore, $OK(F')$ holds.
- Case $P = \mathbf{let } x = \mathbf{malloc in } P'$ and $\langle \mathbf{let } x = \mathbf{malloc in } P', F \rangle \xrightarrow{\mathbf{malloc}(x')} \langle [x'/x]P', F \rangle$
Similiar to above.
- Case $P = \mathbf{let } x = y \mathbf{ in } P'$ and $\langle \mathbf{let } x = y \mathbf{ in } P', F \rangle \rightarrow \langle [x'/x]P', F \rangle$
Similiar to above.
- Case $P = \mathbf{let } x = *y \mathbf{ in } P'$ and $\langle \mathbf{let } x = *y \mathbf{ in } P', F \rangle \rightarrow \langle [x'/x]P', F \rangle$
Similiar to above.
- Case $P = \mathbf{let } x = \mathbf{null in } P'$ and $\langle \mathbf{let } x = \mathbf{null in } P', F \rangle \rightarrow \langle [x'/x]P', F \rangle$
Similiar to above.
- Case $P = \mathbf{free}$ and $\langle \mathbf{free}, F \rangle \xrightarrow{\mathbf{free}} \langle \mathbf{0}, F \rangle$
Similiar to above.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{const}(*x) \notin F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \rangle}$
We need to prove $OK(F)$. From the assumption, $OK(F)$ holds.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{const}(*x) \notin F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \rangle}$
We need to prove $OK(F)$. From the assumption, $OK(F)$ holds.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{null}(*x) \in F \quad \mathbf{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \rangle}$
We need to prove $OK(F)$. From the assumption, $OK(F)$ holds.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\neg \mathbf{null}(*x) \in F \quad \mathbf{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \rangle}$
We need to prove $OK(F)$. From the assumption, it holds.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{null}(*x), \neg \mathbf{null}(*x) \notin F \quad \mathbf{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \cup \mathbf{null}(*x) \rangle}$
We need to prove $OK(F \cup \mathbf{null}(*x))$. From the assumption, we have $OK(F)$ and $\neg \mathbf{null}(*x) \notin F$. Therefore $OK(F \cup \mathbf{null}(*x))$ holds.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{null}(*x), \neg \mathbf{null}(*x) \notin F \quad \mathbf{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \cup \neg \mathbf{null}(*x) \rangle}$
We need to prove $OK(F \cup \neg \mathbf{null}(*x))$. From the assumption, we have $OK(F)$ and $\mathbf{null}(*x) \notin F$. Therefore $OK(F \cup \neg \mathbf{null}(*x))$ holds.

- Case $P = \mathbf{const}(*x)P'$ and $\langle \mathbf{const}(*x)P', F \rangle \rightarrow \langle P'; \mathbf{endconst}(*x), F \cup \{\mathbf{const}(*x)\} \rangle$
We need to prove $OK(F \cup \{\mathbf{const}(*x)\})$. From the assumption, we have $OK(F)$ holds. Also, $F \cup \{\mathbf{const}(*x)\}$ does not contain both $\mathbf{null}(*x)$ and $\neg \mathbf{null}(*x)$. Therefore, $OK(F \cup \{\mathbf{const}(*x)\})$ holds.
- Case $P = \mathbf{endconst}(*x)$ and $\frac{F' = \mathbf{filter}.T(F, *x)}{\langle \mathbf{endconst}(*x), F \rangle \rightarrow \langle \mathbf{0}, F' \rangle}$
we need to prove $OK(F')$. From assumption, we have $OK(F)$ which means F does not contain both $\mathbf{null}(*x)$ and $\neg \mathbf{null}(*x)$. By the definition of *filter* function, we have $F' = F \setminus \{\mathbf{null}(*x), \neg \mathbf{null}(*x)\}$ or $F - \mathbf{const}(*x)$, which means F' does not contain both $\mathbf{null}(*x)$ and $\neg \mathbf{null}(*x)$. Therefore, $OK(F')$ holds.
- Case $P = \mu\alpha.P'$ and $\langle \mu\alpha.P', F \rangle \rightarrow \langle [\mu\alpha.P']P', F \rangle$
We need to prove $OK(F)$. From the assumption, we have that $OK(F)$ holds.
- Case $P = P_1; P_2$ and $\frac{\langle P_1, F \rangle \xrightarrow{\rho} \langle P'_1, F' \rangle}{\langle P_1; P_2, F \rangle \xrightarrow{\rho} \langle P'_1; P_2, F' \rangle}$
We need to prove $OK(F')$. By IH, we have $\langle P_1, F \rangle \xrightarrow{\rho} \langle P'_1, F' \rangle$ and $OK(F)$ holds, then $OK(F')$ holds.

□

Lemma 3.2. *If $OK_n(P, F)$ and $\langle P, F \rangle \xrightarrow{\rho} \langle P', F' \rangle$, then*

- $OK_{n-1}(P', F')$ if $\rho = \mathbf{malloc}$,
- $OK_{n+1}(P', F')$ if $\rho = \mathbf{free}$,
- $OK_n(P', F')$ if $\rho = \text{Otherwise}$

Proof. By induction on $\langle P, F \rangle \xrightarrow{\rho} \langle P', F' \rangle$.

- Case $P = \mathbf{0}; P'$ and $\langle \mathbf{0}; P', F \rangle \rightarrow \langle P', F \rangle$
We need to prove $OK_n(P', F)$. Assume that $OK_n(P', F)$ does not hold. Then, we have (1) $\exists \sigma$ and Q s.t. $\langle P', F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F)$ does not hold.
From the definition of that $OK(\mathbf{0}; P', F)$ holds, we have (1) if $\langle \mathbf{0}; P', F \rangle \rightarrow \langle P', F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F)$, which are in contradiction to the assumption. Therefore, $OK_n(P', F)$ holds.
- Case $P = \mathbf{let } x = \mathbf{malloc} \mathbf{ in } P'$ and $\langle \mathbf{let } x = \mathbf{malloc} \mathbf{ in } P', F \rangle \xrightarrow{\mathbf{malloc}(x')} \langle [x'/x]P', F \rangle$
we need to prove $OK_{n-1}([x'/x]P', F)$. Assume that $OK_{n-1}([x'/x]P', F)$ does not hold. Then we have (1) $\exists \sigma$ and Q s.t. $\langle [x'/x]P', F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m\sigma - \sharp_f\sigma > n$ or (2) $OK(F)$ does not hold.
From the definition of $OK_n(P, F)$, we have (1) $\langle \mathbf{let } x = \mathbf{malloc} \mathbf{ in } P', F \rangle \xrightarrow{\mathbf{malloc}(x')} \langle [x'/x]P', F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n - 1$ and (2) $OK(F)$ holds. Therefore, we get the contradiction, and the $OK_{n-1}([x'/x]P', F)$ holds.
- Case $P = \mathbf{let } x = y \mathbf{ in } P'$ and $\langle \mathbf{let } x = y \mathbf{ in } P', F \rangle \rightarrow \langle [x'/x]P', F \rangle$
Similar to the above.

- Case $P = \text{let } x = *y \text{ in } P'$ and $\langle \text{let } x = *y \text{ in } P', F \rangle \rightarrow \langle [x'/x]P', F \rangle$

Similar to the above.

- Case $P = \text{let } x = \text{null in } P'$ and $\langle \text{let } x = \text{null in } P', F \rangle \rightarrow \langle [x'/x]P', F \rangle$

Similar to the above.

- Case $P = \text{free}$ and $\langle \text{free}, F \rangle \xrightarrow{\text{free}} \langle \mathbf{0}, F \rangle$

We need to prove $OK_{n+1}(\mathbf{0}, F)$, which means we need to prove (1) $\forall \sigma$ and Q if $\langle \mathbf{0}, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F)$ holds. There is no Q and σ s.t. $\langle \mathbf{0}, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, so (1) holds. $OK(F)$ holds from Lemma 3.1. Therefore, $OK(\mathbf{0}, F)$ holds.

- Case $P = \text{endconst}(*x)$ and $\frac{F' = \text{filter}.T(F, *x)}{\langle \text{endconst}(*x), F \rangle \rightarrow \langle \mathbf{0}, F' \rangle}$

We need to prove $OK_n(\mathbf{0}, F')$, which means we need to prove (1) $\forall \sigma$ and Q if $\langle \mathbf{0}, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F')$ holds. There is no Q and σ s.t. $\langle \mathbf{0}, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, so (1) holds. From the assumption $OK_n(P, F)$, we have $OK(F)$, which means F does not contain both $\text{null}(*x)$ and $\neg \text{null}(*x)$. By the definition of function $\text{filter}.T$, we have $F' = F \setminus \{\text{null}(*x), \neg \text{null}(*x)\}$ or $F - \text{const}(*x)$. Therefore $OK(F')$ holds. So $OK_n(\mathbf{0}, F')$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\text{const}(*x) \notin F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \rangle}$

We need to prove $OK_n(P_1, F)$. Assume that $OK_n(P_1, F)$ does not hold. Then, we have (1) $\exists \sigma$ and Q s.t. $\langle P_1, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F)$ does not hold.

From the definition of that $OK_n((*x)(P_1, P_2), F)$ holds, we have (1) if $\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F)$ holds, which are in contradiction to the assumption. Therefore, $OK_n(P_1, F)$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\text{const}(*x) \notin F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \rangle}$

We need to prove $OK_n(P_2, F)$. Assume that $OK_n(P_2, F)$ does not hold. Then, we have (1) $\exists \sigma$ and Q s.t. $\langle P_2, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F)$ does not hold.

From the definition of that $OK_n((*x)(P_1, P_2), F)$ holds, we have (1) if $\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F)$ holds, which are in contradiction to the assumption. Therefore, $OK_n(P_2, F)$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\text{null}(*x) \in F \quad \text{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \rangle}$

We need to prove $OK_n(P_1, F)$. Assume that $OK_n(P_1, F)$ does not hold. Then, we have (1) $\exists \sigma$ and Q s.t. $\langle P_1, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F)$ does not hold.

From the definition of that $OK_n((*x)(P_1, P_2), F)$ holds, we have (1) if $\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F)$ holds, which are in contradiction to the assumption. Therefore, $OK_n(P_1, F)$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\neg \text{null}(*x) \in F \quad \text{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \rangle}$

We need to prove $OK_n(P_2, F)$. Assume that $OK_n(P_2, F)$ does not hold. Then we have (1) $\exists \sigma$ and Q s.t. $\langle P_2, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F)$ does not hold.

From the definition of that $OK_n((*)x)(P_1, P_2, F)$ holds, we have (1) if $\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F)$ holds, which are in contradiction to the assumption. Therefore, $OK_n(P_2, F)$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\text{null}(*x), \neg \text{null}(*x) \notin F \quad \text{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \cup \{\text{null}(*x)\} \rangle}$

We need to prove $OK_n(P_1, F \cup \{\text{null}(*x)\})$. Assume that $OK_n(P_1, F \cup \{\text{null}(*x)\})$ does not hold. Then we have (1) $\exists \sigma$ and Q s.t. $\langle P_1, F \cup \{\text{null}(*x)\} \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F \cup \{\text{null}(*x)\})$ does not hold.

From the definition of that $OK_n((*)x)(P_1, P_2, F)$ holds, we have (1) if $\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_1, F \cup \{\text{null}(*x)\} \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F)$ holds. By $OK(F)$ and $\text{null}(*x), \neg \text{null}(*x) \notin F$, we have $OK(F \cup \{\text{null}(*x)\})$ holds. Therefore, we get the contradiction and $OK_n(P_1, F \cup \{\text{null}(*x)\})$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\text{null}(*x), \neg \text{null}(*x) \notin F \quad \text{const}(*x) \in F}{\langle (*x)(P_1, P_2), F \rangle \rightarrow \langle P_2, F \cup \{\neg \text{null}(*x)\} \rangle}$

Similar to the above.

- Case $P = \text{const}(*x)P'$ and $\langle \text{const}(*x)P', F \rangle \rightarrow \langle P'; \text{endconst}(*x), F \cup \text{const}(*x) \rangle$

We need to prove $OK_n(P'; \text{endconst}(*x), F \cup \text{const}(*x))$. Assume that $OK_n(P'; \text{endconst}(*x), F \cup \text{const}(*x))$ does not hold. Then, we have (1) $\exists \sigma$ and Q s.t. $\langle P'; \text{endconst}(*x), F \cup \text{const}(*x) \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F \cup \text{const}(*x))$ does not hold.

From the definition of that $OK_n(\text{const}(*x)P', F)$ holds, we have (1) if $\langle \text{const}(*x)P', F \rangle \rightarrow \langle P'; \text{endconst}(*x), F \cup \text{const}(*x) \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(F)$ holds, which are in contradiction to the assumption. Therefore, $OK_n(P_1, F)$ holds.

- Case $P = \mu\alpha.P'$ and $\langle \mu\alpha.P', F \rangle \rightarrow \langle [\mu\alpha.P'/\alpha]P', F \rangle$

We need to prove $OK_n([\mu\alpha.P'/\alpha]P', F)$. Assume that $OK_n([\mu\alpha.P'/\alpha]P', F)$ does not hold. Then, we have (1) $\exists \sigma$ and Q s.t. $\langle [\mu\alpha.P'/\alpha]P', F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F)$ does not hold.

From the definition of that $OK_n(\mu\alpha.P', F)$ holds, we have (1) if $\langle \mu\alpha.P', F \rangle \rightarrow \langle [\mu\alpha.P'/\alpha]P', F \rangle \xrightarrow{\sigma} \langle Q, F' \rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$, which is a contradiction; and (2) $OK(F)$ holds. From the Lemma 3.1, $OK(F \cup \neg \text{null}(*x))$ holds. Therefore, $OK([\mu\alpha.P'/\alpha]P', F)$ holds.

- Case $P = P_1; P_2$ and $\frac{\langle P_1, F \rangle \xRightarrow{\rho} \langle P'_1, F' \rangle}{\langle P_1; P_2, F \rangle \xRightarrow{\rho} \langle P'_1; P_2, F' \rangle}$

We need to prove $OK_{n'}(P'_1; P_2, F)$, where n' is determined by

$$n' = \begin{cases} n + 1 & \rho = \text{free} \\ n - 1 & \rho = \text{malloc} \\ n & \text{Otherwise.} \end{cases}$$

Assume that $OK_{n'}(P'_1; P_2, F)$ does not hold. Then, we have (1) $\exists \sigma, Q$ and F'' s.t. $\langle P'_1; P_2, F \rangle \xrightarrow{\sigma} \langle Q, F'' \rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n'$ or (2) $OK(F')$ does not hold.

From the definition of that $OK_n(P_1; P_2, F)$ holds, we have (1) if $\langle P_1; P_2, F \rangle \xRightarrow{\rho} \langle P'_1; P_2, F' \rangle \xrightarrow{\sigma} \langle Q, F'' \rangle$, then $\sharp_m(\rho\sigma) - \sharp_f(\rho\sigma) \leq n$ and (2) $OK(F)$ holds.

From (1), we get $n' + \sharp_m(\rho) - \sharp_f(\rho) < \sharp_m(\rho) + \sharp_m(\sigma) - \sharp_f(\rho) - \sharp_f(\sigma) \leq n$. For any ρ , the $n' + \sharp_m(\rho) - \sharp_f(\rho) = n$, therefore we get a contradiction. By IH, we have $OK(F')$ holds, which is a contradiction. Therefore, $OK_{n'}(P_1; P_2, F')$ holds.

□

Proof of Lemma 2.2: By induction on the derivation of $\langle H, R, s, n, C \rangle \xrightarrow{\rho} \langle H', R', s', n', C' \rangle$.

- Case: $\langle H, R, \mathbf{const}(*x)s, n, C \rangle \rightarrow \langle H, R, s; \mathbf{endconst}(*x), n, C \cup \{\mathbf{const}(*x)\} \rangle$

From the assumption $\Theta; \Gamma \vdash \langle H, R, \mathbf{const}(*x)s, n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma \vdash \mathbf{const}(*x)s : P$ and $OK_n(P, F)$. From the inversion of typing rules, we get $\Theta; \Gamma \vdash s : P''$ and $\mathbf{const}(*x)P'' \leq P$ for some P'' . By subtyping, we have $P''; \mathbf{endconst}(*x) \leq Q$ and $\langle P, F \rangle \Rightarrow \langle Q, F \cup \{\mathbf{const}(*x)\} \rangle$ for some Q .

we need to find P' and F' s.t. $\Theta; \Gamma \vdash s; \mathbf{endconst}(*x) : P'$, $OK_n(P', F')$ and $\langle P, F' \rangle \Rightarrow \langle P', F' \rangle$. Taking Q as P' and $F \cup \{\mathbf{const}(*x)\}$ as F' . Therefore $\langle P, F \rangle \rightarrow \langle P', F' \rangle$ holds, and $OK_n(P', F')$ holds from Lemma 3.2. From $\Theta; \Gamma \vdash s; \mathbf{endconst}(*x) : P''; \mathbf{endconst}(*x)$, $P''; \mathbf{endconst}(*x) \leq Q$ and T-SUB, $\Theta; \Gamma \vdash s; \mathbf{endconst}(*x) : P'$ holds.

- Case: $\langle H, R, \mathbf{endconst}(*x), n, C \rangle \rightarrow \langle H, R, \mathbf{skip}, n, C' \rangle$ where $C' = \mathit{filter_C}(C, *x)$

From the assumption $\Theta; \Gamma \vdash \langle H, R, \mathbf{endconst}(*x), n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma \vdash \mathbf{endconst}(*x) : P$ and $OK_n(P, F)$. From the inversion of typing rules, we get $\Theta; \Gamma \vdash \mathbf{endconst}(*x) : \mathbf{endconst}(*x)$ and $\mathbf{endconst}(*x) \leq P$. By subtyping and function $\mathit{filter_T}(F, *x)$, we get $0 \leq Q$ and $\langle P, F \rangle \rightarrow \langle Q, F'' \rangle$ for some Q .

we need to find P' and F' s.t. $\Theta; \Gamma \vdash \mathbf{skip} : P'$, $OK_n(P', F')$ and $\langle P, F' \rangle \Rightarrow \langle P', F' \rangle$. Taking Q as P' and F'' as F' therefore $F' \approx C'$ from functions $\mathit{filter_T}(F, *x)$ and $\mathit{filter_C}(C, *x)$; $\langle P, F \rangle \rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$ hold. From T-SKIP, T-SUB and $0 \leq Q$, then $\Theta; \Gamma \vdash \mathbf{skip} : P'$ holds.

- Case: $\langle H, R, \mathbf{free}(x), n, C \rangle \xrightarrow{\mathbf{free}} \langle H', R, \mathbf{skip}, n+1, C' \rangle$

From the assumption $\Theta; \Gamma \vdash \langle H, R, \mathbf{free}(x), n, C \rangle : \langle P, F \rangle$, we have $OK_n(P, F)$ and $\Theta; \Gamma \vdash \mathbf{free}(x) : P$. From inversion of the typing rules, we have $\Theta; \Gamma \vdash \mathbf{free}(x) : \mathbf{free}$ and $\mathbf{free} \leq P$. By the subtyping, we have $\langle P, F \rangle \xrightarrow{\mathbf{free}} \langle Q, F \rangle$ and $0 \leq Q$ for some Q .

We need to find P' and F' such that $\langle P, F \rangle \xrightarrow{\mathbf{free}} \langle P', F' \rangle$, $\Theta; \Gamma \vdash \mathbf{skip} : P'$, and $OK_{n+1}(P', F')$. Take Q as P' and F as F' . Then, $\langle P, F \rangle \xrightarrow{\mathbf{free}} \langle P', F' \rangle$ holds, and $OK_{n+1}(P', F')$ holds from Lemma 3.2. We also have $\Theta; \Gamma \vdash \mathbf{skip} : P'$ from T-SKIP, $0 \leq Q$ and T-SUB.

- Case: $\langle H, R, \mathbf{let } x = \mathbf{malloc}() \mathbf{ in } s, n, C \rangle \xrightarrow{\mathbf{malloc}} \langle H', R', [x'/x]s, n-1, C' \rangle$

From the assumption $\Theta; \Gamma \vdash \langle H, R, \mathbf{let } x = \mathbf{malloc}() \mathbf{ in } s, n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma \vdash \mathbf{let } x = \mathbf{malloc}() \mathbf{ in } s : P$ and $OK_n(P, F)$. By the inversion of typing rules, we have $\Theta; \Gamma, x \vdash s : P''$ and $\mathbf{let } x = \mathbf{malloc} \mathbf{ in } P'' \leq P$ for some P'' . By subtyping, we get $\langle P, F \rangle \xrightarrow{\mathbf{malloc}x'} \langle Q, F \rangle$ and $[x'/x]P'' \leq Q$ for some Q .

We need to find P' and F' such that $\Theta; \Gamma, x' \vdash [x'/x]s : P'$ and $\langle P, F \rangle \xrightarrow{\mathbf{malloc}x'} \langle P', F' \rangle$ and $OK_{n-1}(P', F')$. Take Q as P' and F as F' . Then $\langle P, F \rangle \xrightarrow{\mathbf{malloc}x'} \langle P', F' \rangle$ holds, and

$OK_{n-1}(P', F')$ holds by Lemma 3.2. From $\Theta; \Gamma, x \vdash s : P''$ and $\text{let } x = \text{malloc in } P'' \leq P$, we have $\Theta; \Gamma, x'' \vdash [x''/x]s : [x''/x]P''$ and $\text{let } x'' = \text{malloc in } [x''/x]P'' \leq P$, and then by the definition of subtyping we have $[x''/x]P'' \leq Q'$ for some Q' . Therefore, we get $\Theta; \Gamma, x'' \vdash [x''/x]s : Q'$. Take x'' as x' and Q' as P' , then $\Theta; \Gamma, x' \vdash [x'/x]s : P'$ holds.

- Case: $\langle H, R, \text{skip}; s, n, C \rangle \rightarrow \langle H, R, s, n, C \rangle$

From the assumption $\Theta; \Gamma \vdash \langle H, R, \text{skip}; s, n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma \vdash \text{skip}; s : P$ and $OK_n(P, F)$. From the inversion of the typing rules, we get $\Theta; \Gamma \vdash s : P''$ and $0; P'' \leq P$. From the definition of subtyping, we have $\langle P, F \rangle \Rightarrow \langle Q, F \rangle$ and $P'' \leq Q$ for some Q .

We need to find P' and F' such that $\Theta; \Gamma \vdash s : P'$ and $\langle P, F \rangle \rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$. Take Q as P' and F as F' . Then $\langle P, F \rangle \Rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$ hold. We also have $\Theta; \Gamma \vdash s : P'$ from T-SUB, $\Gamma \vdash s : P''$ and $P'' \leq Q$.

- Case: $\langle H, R, *x \leftarrow y, n, C \rangle \rightarrow \langle H', R, \text{skip}, n, C \rangle$

From the assumption $\Theta; \Gamma \vdash \langle H, R, *x \leftarrow y, n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma \vdash *x \leftarrow y : P$ and $OK_n(P, F)$. From the inversion of typing rules, we have $0 \leq P$.

We need to find P' such that $\Theta; \Gamma \vdash \text{skip} : P'$, $\langle P, F \rangle \Rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$. Take P as P' and F as F' . Then, $\langle P, F \rangle \Rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$ hold. We also have $\Theta; \Gamma \vdash \text{skip} : P'$ from T-SKIP, $0 \leq P$ and T-SUB.

- Case: $\langle H, R, \text{let } x = y \text{ in } s, n, C \rangle \rightarrow \langle H, R', [x'/x]s, n, C \rangle$

From the assumption $\Theta; \Gamma \vdash \langle H, R, \text{let } x = y \text{ in } s, n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma, y \vdash \text{let } x = y \text{ in } s : P$ and $OK_n(P, F)$. From the inversion of typing rules, we have $\Theta; \Gamma, x, y \vdash s : P''$ and $\text{let } x = y \text{ in } P'' \leq P$ for some P'' . By subtyping, we have $\langle P, F \rangle \rightarrow \langle Q, F \rangle$ and $[x'/x]P'' \leq Q$ for some Q .

We need to find P' and F' such that $\Theta; \Gamma, x', y \vdash [x'/x]s : P'$, $\langle P, F \rangle \rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$. Take Q as P' and F as F' . Then $\langle P, F \rangle \Rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$ hold. From $\Theta; \Gamma, x, y \vdash s : P''$ and $\text{let } x = y \text{ in } P'' \leq P$, we have $\Theta; \Gamma, x'', y \vdash [x''/x]s : [x''/x]P''$ and $\text{let } x'' = y \text{ in } [x''/x]P'' \leq P$, and then by subtyping we have $[x''/x]P'' \leq Q'$ for some Q' . Therefore, we have $\Theta; \Gamma, x'', y \vdash [x''/x]s : Q'$. Take x'' as x' and Q' as P' , then $\Theta; \Gamma, x', y \vdash [x'/x]s : P'$ holds.

- Case: $\langle H, R, \text{let } x = \text{null in } s, n \rangle \rightarrow \langle H, R', [x'/x]s, n \rangle$

Similar to the above.

- Case: $\langle H, R, \text{let } x = *y \text{ in } s, n \rangle \rightarrow \langle H, R', [x'/x]s, n \rangle$

Similar to the above.

- Case: $\langle H, R, \text{ifnull } (*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle \rightarrow \langle H, R, s_1, n, C \rangle$ if $H(R(x)) = \text{null}$ and $\text{const}(*x) \notin C$

From assumption $\Theta; \Gamma \vdash \langle H, R, \text{ifnull } (*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma \vdash \text{ifnull } (*x) \text{ then } s_1 \text{ else } s_2 : P$ and $OK_n(P, F)$. From the inversion of typing rules, we have $\Theta; \Gamma \vdash s_1 : P_1$, $\Theta; \Gamma \vdash s_2 : P_2$ and $(*x)(P_1, P_2) \leq P$. By subtyping and $\text{const}(*x) \notin C$, which means $\text{const}(*x) \notin F$, we get $\langle P, F \rangle \Rightarrow \langle Q, F \rangle$ and $P_1 \leq Q$ for some Q .

We need to find P' and F' such that $\Theta; \Gamma \vdash s_1 : P'$, $\langle P, F \rangle \Longrightarrow \langle P', F' \rangle$ and $OK_n(P', F')$. Take Q as P' and F as F' . Then $\langle P, F \rangle \rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$ hold. We also have $\Theta; \Gamma \vdash s_1 : P'$ from T-SUB, $\Theta; \Gamma \vdash s_1 : P_1$ and $P_1 \leq Q$.

- Case: $\langle H, R, \text{ifnull } (*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle \rightarrow \langle H, R, s_1, n, C \rangle$ if $H(R(x)) \neq \text{null}$ and $\text{const}(*x) \notin C$

Similar to the above.

- Case: $\langle H, R, \text{ifnull } (*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle \rightarrow \langle H, R, s_1, n, C' \rangle$ if $H(R(x)) = \text{null}$, $\text{const}(*x) \in C$ and $C' = C \cup \{\text{null}(*x)\}$

From assumption $\Theta; \Gamma \vdash \langle H, R, \text{ifnull } (*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma \vdash \text{ifnull } (*x) \text{ then } s_1 \text{ else } s_2 : P$ and $OK_n(P, F)$. From the inversion of typing rules, we have $\Theta; \Gamma \vdash s_1 : P_1$, $\Theta; \Gamma \vdash s_2 : P_2$ and $(*) (P_1, P_2) \leq P$. By subtyping, $\text{const}(*x) \in C$ and $\text{assume}(*x \neq \text{null}) \notin C$ which are similar to $\text{const}(*x) \in F$ and $\neg \text{null}(*x) \notin F$, we get $\langle P, F \rangle \Longrightarrow \langle Q, F \cup \{\text{null}(*x)\} \rangle$ and $P_1 \leq Q$ for some Q .

We need to find P' and F' such that $\Theta; \Gamma \vdash s_1 : P'$, $\langle P, F \rangle \Longrightarrow \langle P', F' \rangle$ and $OK_n(P', F')$. Take Q as P' and $F \cup \{\text{null}(*x)\}$ as F' . Then $C' \approx F'$, $\langle P, F \rangle \rightarrow \langle P', F' \rangle$ and $OK_n(P', F')$ hold. We also have $\Theta; \Gamma \vdash s_1 : P'$ from T-SUB, $\Theta; \Gamma \vdash s_1 : P_1$ and $P_1 \leq Q$.

- Case: $\langle H, R, \text{ifnull } (*x) \text{ then } s_1 \text{ else } s_2, n, C \rangle \rightarrow \langle H, R, s_2, n, C' \rangle$ if $H(R(x)) \neq \text{null}$, $\text{const}(*x) \in C$ and $C' = C \cup \{\neg \text{null}(*x)\}$

Similar to the above proof.

- Case: $\langle H, R, s_1; s_2, n, C \rangle \rightarrow \langle H', R', s'_1; s_2, n', C' \rangle$

From the assumption $\Theta; \Gamma \vdash \langle H, R, s_1; s_2, n, C \rangle : \langle P, F \rangle$, we have $\Theta; \Gamma \vdash s_1; s_2 : P$ and $OK_n(P, F)$ with $C \approx F$. By inversion of typing rules, we have $\Theta; \Gamma \vdash s_1 : P_1$, $\Theta; \Gamma \vdash s_2 : P_2$ and $P_1; P_2 \leq P$ for some P_1 and P_2 .

By IH on $\langle H, R, s_1, n, C \rangle$ with derivation $\langle H, R, s_1, n, C \rangle \xrightarrow{\rho} \langle H', R', s'_1, n', C' \rangle$, we have $\exists P'_1, F'_1$ s.t. $\Theta; \Gamma \vdash \langle H', R', s'_1, n', C' \rangle : \langle P'_1, F'_1 \rangle$ and $\langle P_1, F \rangle \xrightarrow{\rho} \langle P'_1, F'_1 \rangle$.

By subtyping we have $\langle P, F \rangle \xrightarrow{\rho} \langle Q, F'_1 \rangle$ and $P'_1; P_2 \leq Q$ for some Q .

We need to find P' and F' s.t. $\langle P, F \rangle \xrightarrow{\rho} \langle P', F' \rangle$, $OK_n(P', F')$ and $\Theta; \Gamma \vdash s'_1; s_2 : P'$. Take Q as P' and F'_1 as F' , $\langle P, F \rangle \xrightarrow{\rho} \langle P', F' \rangle$ and $OK_n(P', F')$ hold. By T-Sub, $\Theta; \Gamma \vdash s'_1; s_2 : P'_1; P_2$ and $P'_1; P_2 \leq Q$, we have $\Theta; \Gamma \vdash s'_1; s_2 : P'$ holds.

□

We write $\langle H, R, s, n, C \rangle \xrightarrow{\rho}$ if there is a transition $\xrightarrow{\rho}$ from $\langle H, R, s, n, C \rangle$.

Lemma 3.3. *If $\Theta; \Gamma \vdash \langle H, R, s, n, C \rangle : \langle P, F \rangle$ and $\langle H, R, s, n, C \rangle \xrightarrow{\rho}$ and $\rho \in \{\text{malloc}, \text{free}\}$, then there exists P' and F' such that $\langle P, F \rangle \xrightarrow{\rho} \langle P', F' \rangle$.*

Proof. Induction on the derivation of $\Theta; \Gamma \vdash \langle H, R, s, n, C \rangle : \langle P, F \rangle$. □

Proof of Lemma 2.3:

By contradiction. Assume $\langle H, R, s, n, C \rangle \xrightarrow{\rho} \text{OutOfMemory}$. Then, n is 0 and $\rho = \text{malloc}$ from SEM-OUTOFMEM. From the assumption we have $\Theta; \Gamma \vdash s : P$ and $OK_0(P, F)$. From

Lemma 3.3, there exists P' and F' such that $\langle P, F \rangle \xRightarrow{\mathbf{malloc}} \langle P', F' \rangle$. However, this contradicts $OK_0(P, F)$. □

Proof of Theorem 2.1:

We have $\Theta; \emptyset \vdash s:P, \vdash D:\Theta$ and $OK_n(P, F)$.

Suppose that there exists σ such that $\langle \emptyset, \emptyset, s, n, C \rangle \xrightarrow{\sigma} \langle H', R', s', n', C' \rangle \xrightarrow{\rho} \mathbf{OutOfMemory}$. Then, $n' = 0$ and $\rho = \mathbf{malloc}$. From Lemma 2.2, there exists P' and F' such that $\Theta; \Gamma' \vdash s':P'$, $\langle P, F \rangle \xRightarrow{\sigma} \langle P', F' \rangle$, and $OK_0(P', F')$; hence $\langle H', R', s', 0 \rangle \xrightarrow{\mathbf{malloc}}$. However, this contradicts Lemma 2.3. □