An Extended Behavioral Type System for Memory-Leak Freedom

November 10, 2016

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.

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

Notation $\vec{x}$ is for a finite sequence $\{x_1, ..., x_n\}$, 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 **Var** is a countably infinite set of *variables* and each variable is a pointer. The statement **skip** means "does nothing". The statement $s_1; s_2$ is a sequential execution of s_1 and s_2. The statement $*x \leftarrow y$ changes the content of cell which is pointed to by x with the value y. The statement **free**(x) deallocates a memory cell which is pointed to by pointer x. The statement **let** $x = e$ **in** s evaluates the expression e, binds x to the result, and executes s. The expression **malloc**() allocates a new memory cell. The expression **null** evaluates to the null pointer. The expression $*y$ means dereferencing a memory cell pointed to by y. The statement **ifnull** $(*x)$**then** s_1**else** s_2 executes s_1 if $*x$ is **null** and executes s_2 otherwise. The statement $f(\vec{x})$ expresses a procedure f with arguments $\vec{x}$. The statement **const**$(*x)s$ means $(*x)$ is a constant in statement s. The statement **endconst**$(*x)$ means from this point $(*x)$ maybe not a 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 $\langle d_1 \cup \ldots d_n \rangle$, 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, which can be formalized to check memory leaks even for nonterminating programs;
- C is a set related to current constant pointers, which contains $\mathbf{const}(*x)$, $\mathbf{null}(*x)$ 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 an action, which is as follows.

$$\rho \text{ (label)} \quad ::= \quad \mathbf{malloc} \mid \mathbf{free} \mid \mathbf{null}(*x) \mid \neg\mathbf{null}(*x) \mid \tau$$

The action $\mathbf{malloc}(x')$ expresses an allocation of a new memory cell, and the new cell binds to a fresh variable x'; **free** expresses a deallocation of a memory cell; $\mathbf{null}(*x)$ means $*x$ is a null pointer, and $\neg\mathbf{null}(*x)$ not; τ expresses the other internal actions. For the operational semantics, we often omit τ in $\xrightarrow{\tau}_D$. The metavariable σ is used for a finite sequence of actions $\rho_1 \ldots \rho_n$. The $\xrightarrow{\rho_1 \cdots \rho_n}_D$ is short for $\xrightarrow{\rho_1}_D \xrightarrow{\rho_2}_D \ldots \xrightarrow{\rho_n}_D$. The $\xRightarrow{\rho}_D$ means $\longrightarrow^*_D \xrightarrow{\rho}_D \longrightarrow^*_D$. We write $\xRightarrow{\rho_1 \cdots \rho_n}_D$ for $\xRightarrow{\rho_1}_D \ldots \xRightarrow{\rho_n}_D$.

Notation the $\mathbf{Dom}(f)$ is a mapping from function name f to its domain; for a map f, the $f\{x \mapsto v\}$ and $f \backslash x$ are defined as follows:

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

and $filter(C, *x)$ is defined by a pseudocode as follows:

$$\begin{aligned} filter(C, *x) \quad = \quad & let\ C' = C - \mathbf{const}(*x)\ in \\ & if\ \mathbf{const}(*x) \in\ C'\ then\ return\ C' \\ & else\ return\ C' \backslash \{\mathbf{null}(*x), \neg\mathbf{null}(*x)\} \end{aligned}$$

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

- SEM-CONSTSEQ: **const**$(*x)$ and **endconst**$(*x)$ together guarantees a pointer pointed to by $*x$ cannot be changed in the statement s. The set C with the new added **const**$(*x)$ describes this status.

- SEM-IFNULLT and SEM-IFCONSTNULLT: these two rules represents if $(*x)$ is a null pointer, the statement s_1 will be executed. the difference of the two is if the **const**$(*x)$ is in set C then**null**$(*x)$ is added to C, which means $(*x)$ is a null pointer and cannot be updated from now on; otherwise $(*x)$ can be changed, like SEM-IFNULLT.

- SEM-FREE: deallocating one memory cell pointed by x is to remove linkage of pointer variable x to heap; this action will release one memory cell space, which increments the number of available memory cells n by one.

- SEM-MALLOC and SEM-OUTOFMEM: allocating one memory cell is described as updating the heap by adding a fresh heap variable l to anywhere v of the heap and adding the linkage of a fresh register variable x' to that l; This action is allowed only if the number of available memory cells is positive; otherwise **OutOfMemory**.

- SEM-ASSIGNEXN, SEM-FREEEXN and SEM-DEREFEXN: these rules express that accessing a null pointer or a dangling pointer will give raise to an exceptional state **MemEx**. However, in this paper we do not see the state **MemEx** is an erroneous state, hence a well-typed program may lead to these states. One thing we should notice the command **free**(x) , if x is a null pointer, raises state **MemEx** in the current semantics, although it is equivalent to **skip** in the C language.

- SEM-ASSIGNCONSTEXN: expressing that if a constant memory cell pointed to by x or its aliases are changed it will raise exceptional state **ConstEx**.

In order to deal with a path-sensitive program to guarantee *total* memory-leak freedom, we redefined the several definitions as follows. 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* $\langle H, R, s, n, C\rangle \stackrel{\sigma}{\Longrightarrow}$ **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.

$$
\begin{array}{llcl}
P & \text{(behavioral types)} & ::= & \mathbf{0} \mid P_1; P_2 \mid \mathbf{malloc} \mid \mathbf{free} \mid \alpha \mid \mu\alpha.P \\
 & & & \mid (x)P \mid (*x)(P_1, P_2) \mid \mathbf{const}(*x)P \mid \mathbf{endconst}(*x) \\
\Gamma & \text{(variable type environment)} & ::= & \{x_1, x_2, \ldots, x_n\} \\
\Psi & \text{(dependent function type)} & ::= & (\vec{x})P \\
\Theta & \text{(function type environment)} & ::= & \{f_1 : \Psi_1, \ldots, f_n : \Psi_n\}
\end{array}
$$

$$\frac{C' = filter(C, *x)}{\langle H, R, \textbf{endconst}(*x), n, C\rangle \to_D \langle H, R, \textbf{skip}, n, C'\rangle} \quad (\textsc{Sem-ConstSkip})$$

$$\langle H, R, \textbf{const}(*x)s, n, C\rangle \to_D \langle H, R, s; \textbf{endconst}(*x), n, C \cup \{\textbf{const}(*x)\}\rangle \quad (\textsc{Sem-ConstSeq})$$

$$\langle H, R, \textbf{skip}; s, n, C\rangle \longrightarrow_D \langle H, R, s, n, C\rangle \quad (\textsc{Sem-Skip})$$

$$\frac{\langle H, R, s_1, n, C\rangle \xrightarrow{\rho}_D \langle H', R', s_1', n', C'\rangle}{\langle H, R, s_1; s_2, n, C\rangle \xrightarrow{\rho}_D \langle H', R', s_1'; s_2, n', C'\rangle} \quad (\textsc{Sem-Seq})$$

$$\frac{x' \notin \textbf{Dom}(R)}{\langle H,\ R,\ \textbf{let}\ x = \textbf{null}\ \textbf{in}\ s, n, C\rangle \longrightarrow_D \langle H,\ R\{x' \mapsto \textbf{null}\},\ [x'/x]\, s, n, C\rangle} \quad (\textsc{Sem-LetNull})$$

$$\frac{x' \notin \textbf{Dom}(R)}{\langle H,\ R,\ \textbf{let}\ x = y\ \textbf{in}\ s, n, C\rangle \longrightarrow_D \langle H,\ R\{x' \mapsto R(y)\},\ [x'/x]\, s, n, C\rangle} \quad (\textsc{Sem-LetEq})$$

$$\frac{H(R(x)) = \textbf{null}, \textbf{const}(*x) \notin C}{\langle H,\ R,\ \textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,\ n, C\rangle \xrightarrow{\textbf{null}(*x)}_D \langle H,\ R,\ s_1,\ n, C\rangle} \quad (\textsc{Sem-IfNullT})$$

$$\frac{H(R(x)) \neq \textbf{null}, \textbf{const}(*x) \notin C}{\langle H,\ R,\ \textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,\ n, C\rangle \xrightarrow{\neg\textbf{null}(*x)}_D \langle H,\ R,\ s_2,\ n, C\rangle} \quad (\textsc{Sem-IfNullF})$$

$$\frac{H(R(x)) = \textbf{null}, \textbf{const}(*x) \in C}{\langle H,\ R,\ \textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,\ n, C\rangle \xrightarrow{\textbf{null}(*x)}_D \langle H,\ R,\ s_1,\ n, C \cup \{\textbf{null}(*x)\}\rangle} \quad (\textsc{Sem-IfConstNullT})$$

$$\frac{H(R(x)) \neq \textbf{null}, \textbf{const}(*x) \in C}{\langle H,\ R,\ \textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,\ n, C\rangle \xrightarrow{\neg\textbf{null}(*x)}_D \langle H,\ R,\ s_2,\ n, C \cup \{\neg\textbf{null}(*x)\}\rangle} \quad (\textsc{Sem-IfConstNullF})$$

$$\frac{\forall z. R(x) = R(z) \Rightarrow \textbf{const}(*x) \notin C}{\langle H\{R(x) \mapsto v\}, R, *x \leftarrow y, n, C\rangle \longrightarrow_D \langle H\{R(x) \mapsto R(y)\}, R, \textbf{skip}, n, C\rangle} \quad (\textsc{Sem-Assign})$$

$$\frac{x' \notin \textbf{Dom}(R) \qquad R(y) \in \textbf{Dom}(H)}{\langle H,\ R,\ \textbf{let}\ x = *y\ \textbf{in}\ s, n, C\rangle \longrightarrow_D \langle H,\ R\{x' \mapsto H(R(y))\},\ [x'/x]\, s, n, C\rangle} \quad (\textsc{Sem-LetDeref})$$

$$\frac{R(x) \neq \textbf{null}\ \text{and}\ R(x) \in \textbf{Dom}(H)}{\langle H\{R(x) \mapsto v\},\ R,\ \textbf{free}(x), n, C\rangle \xrightarrow{\textbf{free}}_D \langle H \backslash R(x),\ R,\ \textbf{skip}, n+1, C\rangle} \quad (\textsc{Sem-Free})$$

$$\frac{l \notin \textbf{Dom}(H) \qquad n > 0}{\langle H,\ R,\ \textbf{let}\ x = \textbf{malloc}()\ \textbf{in}\ s, n, C\rangle \xrightarrow{\textbf{malloc}}_D \langle H\{l \mapsto v\},\ R\{x' \mapsto l\},\ [x'/x]\, s, n-1, C\rangle} \quad (\textsc{Sem-Malloc})$$

$$\frac{D(f) = (\vec{y})s}{\langle H,\ R,\ f(\vec{x}), n, C\rangle \longrightarrow_D \langle H,\ R,\ [\vec{x}/\vec{y}]\, s, n, C\rangle} \quad (\textsc{Sem-Call})$$

$$\frac{R(x) = \textbf{null}\ \text{or}\ R(x) \notin \textbf{Dom}(H)}{\langle H,\ R,\ \textbf{free}(x), n, C\rangle \xrightarrow{\textbf{free}}_D \textbf{MemEx}} \quad (\textsc{Sem-FreeExn})$$

$$\frac{R(x) = \textbf{null}\ \text{or}\ R(x) \notin \textbf{Dom}(H)}{\langle H,\ R,\ *x \leftarrow y, n, C\rangle \longrightarrow_D \textbf{MemEx}} \quad (\textsc{Sem-AssignExn})$$

$$\frac{R(y) = \textbf{null}\ \text{or}\ R(y) \notin \textbf{Dom}(H)}{\langle H,\ R,\ \textbf{let}\ x = *y\ \textbf{in}\ s, n, C\rangle \longrightarrow_D \textbf{MemEx}} \quad (\textsc{Sem-DerefExn})$$

$$\frac{\exists z. \textbf{const}(*z) \in C\ \text{and}\ R(x) = R(z)}{\langle H\{R(x) \mapsto v\}, R, *x \leftarrow y, n, C\rangle \longrightarrow_D \textbf{ConstEx}} \quad (\textsc{Sem-AssignConstExn})$$

$$\langle H,\ R,\ \textbf{let}\ x = \textbf{malloc}()\ \textbf{in}\ s, 0, C\rangle \xrightarrow{\textbf{malloc}}_D \textbf{OutOfMemory} \quad (\textsc{Sem-OutOfMem})$$

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

Behavioral types ranged over by P express the abstaction of behaviors of a program. The type $\mathbf{0}$ represents the does nothing behavior; the type $P_1; P_2$ describes a sequential execution of behavioral type P_1 and P_2; The type **malloc** expresses an allocation of a memory cell; the type **free** represents a deallocation of a pointer; the type $\mu\alpha.P$ represents a recursive substitution of α in P ; the type $(*x)(P_1, P_2)$ represents that P_1 or P_2 is obtained dependent on $*x$, e.g., P_1 is obtained if $*x$ is not a null pointer, otherwise P_2; 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 value in 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 without information about their types, because our focus is the behavior of a program.

Dependent function types ranged over by Ψ represents the behavior of a function. $\vec{x}$ is the formal arguments of the function, and the behavioral type P obtained dependent on $\vec{x}$.

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

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

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 $OK_n(P, C)$, a predicate, where P represents the behavior of a program which consumes at most n memory cells under constant value environment C.

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

Definition 4. *$OK_n(P, C)$ holds if $\forall P'$ and σ. if $\langle P, C\rangle \xrightarrow{\sigma} \langle P', C'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$*

Intuitively, $OK_n(P, C)$ represents at very running steps, the number of memory cells a program consumed will not exceed the number of memory cells the program requires.

Definition 5 (Subtyping)**.** *$C \vdash P_1 \leq P_2$ is the largest relation such that, for any P_1', C' and ρ, if $\langle P_1, C\rangle \xrightarrow{\rho} \langle P_1', C'\rangle$, then there exists P_2' such that $\langle P_2, C\rangle \xRightarrow{\rho} \langle P_2',' \rangle$ and $C' \vdash P_1' \leq P_2'$. We write $P_1 \leq P_2$ if $C \vdash P_1 \leq P_2$ for any C.*

Figure 3 shows the typing rules. For example, the rule T-IfNull represents the behavior of **ifnull** $(*x)$ **then** s_1 **else** s_2 is abstracted as $(*x)(P_1, P_2)$ where P_1 and P_2 are the behavior of s_1 and s_2 respectively; this conditional statement means that executing s_1 if $(*x)$ is a null pointer, otherwise s_2. The typing rule T-Program represents a program requires at most n memory cells during running under the predication $OK_n(P, C)$, where P is behavioral type of statement s.

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.

$$\langle \mathbf{0}; P, C\rangle \rightarrow \langle P, C\rangle \qquad \text{(Tr-Skip)}$$

$$\langle \mathbf{free}, C\rangle \xrightarrow{\mathbf{free}} \langle \mathbf{0}, C\rangle \quad \text{(Tr-Free)} \qquad \langle \mu\alpha.P, C\rangle \rightarrow \langle [\mu\alpha.P/\alpha]P, C\rangle \text{ (Tr-Rec)}$$

$$\langle P_1 + P_2, C\rangle \rightarrow \langle P_1, C\rangle \text{ (Tr-ChoiceL)} \qquad \langle P_1 + P_2, C\rangle \rightarrow \langle P_2, C\rangle \text{ (Tr-ChoiceR)}$$

$$\frac{\langle P_1, C\rangle \xrightarrow{\rho} \langle P_1', C'\rangle}{\langle P_1; P_2, C\rangle \xrightarrow{\rho} \langle P_1'; P_2, C'\rangle} \qquad \text{(Tr-Seq)}$$

$$\langle \mathbf{malloc}, C\rangle \xrightarrow{\mathbf{malloc}} \langle 0, C\rangle \qquad \text{(Tr-Malloc)}$$

$$\frac{x' is fresh}{\langle (x)P, C\rangle \rightarrow \langle [x'/x]P, C\rangle} \qquad \text{(Tr-Bind)}$$

$$\langle \mathbf{const}(*x)P, C\rangle \rightarrow \langle P; \mathbf{endconst}(*x), C \cup \{\mathbf{const}(*x)\}\rangle \qquad \text{(Tr-Const)}$$

$$\frac{C' = filter(C, *x)}{\langle \mathbf{endconst}(*x), C\rangle \rightarrow \langle \mathbf{0}, C'\rangle} \qquad \text{(Tr-Endconst)}$$

$$\frac{\mathbf{const}(*x) \notin C}{\langle (*x)(P_1, P_2), C\rangle \xrightarrow{\mathbf{null}(*x)} \langle P_1, C\rangle} \text{ (Tr-NotConst1)} \qquad \frac{\mathbf{const}(*x) \notin}{\langle (*x)(P_1, P_2), C\rangle \xrightarrow{\neg\mathbf{null}(*x)} \langle P_2, \rangle} \text{ (Tr-NotConst2)}$$

$$\frac{\mathbf{null}(*x) \in C \qquad \mathbf{const}(*x) \in C}{\langle (*x)(P_1, P_2), C\rangle \rightarrow \langle P_1, C\rangle} \text{ (Tr-NullIn)} \qquad \frac{\neg\mathbf{null}(*x) \in C \qquad \mathbf{const}(*x) \in C}{\langle (*x)(P_1, P_2), C\rangle \rightarrow \langle P_2, C\rangle} \text{ (Tr-NNullIn)}$$

$$\frac{\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin C \qquad \mathbf{const}(*x) \in}{\langle (*x)(P_1, P_2), C\rangle \xrightarrow{\mathbf{null}(*x)} \langle P_1, C \cup \mathbf{null}(*x)\rangle} \qquad \text{(Tr-NNullNotIn1)}$$

$$\frac{\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin C \qquad \mathbf{const}(*x) \in C}{\langle (*x)(P_1, P_2), C\rangle \xrightarrow{\neg\mathbf{null}(*x)} \langle P_2, C \cup \neg\mathbf{null}(*x)\rangle} \qquad \text{(Tr-NNullNotIn2)}$$

Figure 2: semantics of behavioral types with dependent types.

$$\Theta; \Gamma \vdash \mathbf{skip} : \mathbf{0} \qquad \text{(T-Skip)}$$

$$\frac{\Theta; \Gamma \vdash s_1 : P_1 \quad \Theta; \Gamma \vdash s_2 : P_2}{\Theta; \Gamma \vdash s_1; s_2 : P_1; P_2} \ \text{(T-Seq)}$$

$$\Theta; \Gamma, x, y \vdash *x \leftarrow y : \mathbf{0} \quad \text{(T-Assign)}$$

$$\Theta; \Gamma, x \vdash \mathbf{free}(x) : \mathbf{free} \qquad \text{(T-Free)}$$

$$\frac{\Theta; \Gamma, x \vdash s : P}{\Theta; \Gamma \vdash \mathbf{let}\ x = \mathbf{malloc}()\ \ \mathbf{in}\ s : \mathbf{malloc}\ (x)P} \ \text{(T-Malloc)}$$

$$\frac{\Theta; \Gamma, x, y \vdash s : P}{\Theta; \Gamma, y \vdash \mathbf{let}\ x = y\ \mathbf{in}\ s : [y/x]P} \ \text{(T-LetEq)}$$

$$\frac{\Theta; \Gamma, x, y \vdash s : P}{\Theta; \Gamma, y \vdash \mathbf{let}\ x = *y\ \mathbf{in}\ s : (x)P} \ \text{(T-LetDeref)}$$

$$\frac{\Theta; \Gamma, x \vdash s : P}{\Theta; \Gamma \vdash \mathbf{let}\ x = \mathbf{null}\ \mathbf{in}\ s : (x)P} \ \text{(T-LetNull)}$$

$$\Theta; \Gamma, x \vdash \mathbf{endconst}(*x) : \mathbf{endconst}(*x) \qquad \text{(T-Endconst)}$$

$$\frac{\Theta; \Gamma, x \vdash s : P}{\Theta; \Gamma, x \vdash \mathbf{const}(*x)s : \mathbf{const}(*x)P} \qquad \text{(T-Const)}$$

$$\frac{\Theta; \Gamma, x \vdash s_1 : P_1 \qquad \Theta; \Gamma, x \vdash s_2 : P_2}{\Theta; \Gamma, x \vdash \mathbf{ifnull}\ (*x)\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2 : (*x)(P_1, P_2)} \qquad \text{(T-IfNull)}$$

$$\Theta, f : (\vec{y})P; \Gamma, \vec{x} \vdash f(\vec{x}) : P[\vec{x}/\vec{y}] \qquad \text{(T-Call)}$$

$$\frac{\Theta; \Gamma \vdash s : P_1 \qquad P_1 \leq P_2}{\Theta; \Gamma \vdash s : P_2} \qquad \text{(T-Sub)}$$

$$\frac{\Theta(f) = (\vec{x})P \qquad \mathbf{Dom}(D) = \mathbf{Dom}(\Theta) \qquad \Theta; x_1, \ldots, x_n \vdash s : P \text{ for each } f \mapsto (x_1, \ldots, x_n)s \in D}{\vdash D : \Theta} \ \text{(T-Def)}$$

$$\frac{\vdash D : \Theta \qquad \Theta; \emptyset \vdash s : P \quad OK_n(P, C)}{\vdash \langle D, s \rangle : n} \qquad \text{(T-Program)}$$

Figure 3: typing rules

Definition 6. *consistency*(H, R, C)*: for all x such that (1) C does not contain both* $\neg\textbf{null}(*x)$ *and* $\textbf{null}(*x)$*. (2) if* $\textbf{const}(*x) \in C$ *and* $\textbf{null}(*x) \in C$*, then* $H(R(x)) = null$*. (3) if* $\textbf{const}(*x) \in C$ *and* $\neg\textbf{null}(*x) \in C$*, then* $H(R(x)) \neq null$*.*

Definition 7. *we write* $\Theta \vdash \langle H, R, s, n, C\rangle : \langle P, C\rangle$*, if there exists* Γ *such that* $\Theta; \Gamma \vdash s : P$*,* $OK_n(P, C)$*, consistency*(H, R, C) *and* $\Gamma \subseteq \textbf{Dom}(R)$*.*

Lemma 2.2 (Preservation)**.** *suppose that* $\Theta \vdash \langle H, R, s, n, C\rangle : \langle P, C\rangle$*, if* $\langle H, R, s, n, C\rangle \xrightarrow{\rho} \langle H', R', s', n', C'\rangle$ *then* $\exists P'$ *and* C' *s.t. (1)* $\Theta \vdash \langle H', R', s', n', C'\rangle : \langle P', C'\rangle$ *and (2)* $\langle P, C\rangle \overset{\rho}{\Longrightarrow} \langle P', C'\rangle$*.*

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

3 Proof of Lemmas

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

- $OK_{n-1}(P', C')$ *if* $\rho = \textbf{malloc}$*,*
- $OK_{n+1}(P', C')$ *if* $\rho = \textbf{free}$*,*
- $OK_n(P', C')$ *if* $\rho = Otherwise$

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

- Case $P = \textbf{0}; P'$ and $\langle \textbf{0}; P', C\rangle \to \langle P', C\rangle$

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

  From the definition of that $OK_n(\textbf{0}; P', C)$ holds, we have (1) if $\langle \textbf{0}; P', C\rangle \to \langle P', C\rangle \xrightarrow{\sigma} \langle Q, C'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$, which are in contradiction to the assumption $\sharp_m(\sigma) - \sharp_f(\sigma) > n$. Therefore, $OK_n(P', C)$ holds.

- Case $P = \textbf{malloc}$ and $\textbf{malloc}, C\rangle \xrightarrow{\textbf{malloc}} \langle 0, C\rangle$

  we need to prove $OK_{n-1}(0, C)$, which means we need to prove that for all σ and Q, if $\langle 0, C\rangle \xrightarrow{\sigma} \langle Q, C'\rangle$ then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n - 1$. There is no σ and Q such that $\langle 0, C\rangle \xrightarrow{\sigma} \langle Q, C'\rangle$. Therefore, $OK_{n-1}(0, C)$ holds.

- Case $P = \textbf{let}\ x = y\ \textbf{in}\ P'$ and $\langle \textbf{let}\ x = y\ \textbf{in}\ P', C\rangle \to \langle [x'/x]P', C\rangle$

  [TODO]

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

  Similar to the above.

- Case $P = \textbf{let}\ x = \textbf{null in}\ P'$ and $\langle \textbf{let}\ x = \textbf{null in}\ P', C\rangle \to \langle [x'/x]P', C\rangle$

  Similar to the above.

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

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

- Case $P = \mathbf{endconst}(*x)$ and $\frac{C'=filter(C,*x)}{\langle \mathbf{endconst}(*x), C\rangle \rightarrow \langle \mathbf{0}, C'\rangle}$

  We need to prove $OK_n(\mathbf{0}, C')$, which means we need to prove $\forall \sigma$ and Q if $\langle \mathbf{0}, C\rangle \xrightarrow{\sigma} \langle Q, C'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ and (2) $OK(C')$ holds. There is no Q and σ s.t. $\langle \mathbf{0}, C\rangle \xrightarrow{\sigma} \langle Q, C\rangle$. So $OK_n(\mathbf{0}, C')$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{const}(*x) \notin C}{\langle (*x)(P_1,P_2), C\rangle \xrightarrow{\mathbf{null}(*x)} \langle P_1, C\rangle}$

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

  From the definition of that $OK_n((*x)(P_1, P_2), C)$ holds, we have (1) if $\langle (*x)(P_1, P_2), C\rangle \xrightarrow{\mathbf{null}(*x)} \langle P_1, C\rangle \xrightarrow{\sigma} \langle Q, C'\rangle$ then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$, which is in contradiction to the assumption $\sharp_m(\sigma) - \sharp_f(\sigma) > n$. Therefore, $OK_n(P_1, C)$ holds.

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

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

  From the definition of that $OK_n((*x)(P_1, P_2), C)$ holds, we have if $\langle (*x)(P_1, P_2), C\rangle \xrightarrow{\neg\mathbf{null}(*x)} \langle P_2, C\rangle \xrightarrow{\sigma} \langle Q, C'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$, which is in contradiction to the assumption. Therefore, $OK_n(P_2, C)$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{null}(*x) \in C \qquad \mathbf{const}(*x) \in C}{\langle (*x)(P_1,P_2), C\rangle \rightarrow \langle P_1, C\rangle}$

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

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

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

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

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

- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin C \qquad \mathbf{const}(*x) \in C}{\langle (*x)(P_1,P_2), C\rangle \xrightarrow{\mathbf{null}(*x)} \langle P_1, C \cup \{\mathbf{null}(*x)\}\rangle}$

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

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

- Case $P = (*x)(P_1, P_2)$ and $\dfrac{\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin C \qquad \mathbf{const}(*x) \in C}{\langle (*x)(P_1,P_2), C\rangle \xrightarrow{\neg\mathbf{null}(*x)} \langle P_2, C \cup \{\neg\mathbf{null}(*x)\}\rangle}$

  Similar to the above.

- Case $P = \mathbf{const}(*x)P'$ and $\langle \mathbf{const}(*x)P', C\rangle \to \langle P'; \mathbf{endconst}(*x), C \cup \mathbf{const}(*x)\rangle$

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

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

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

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

  From the definition of that $OK_n(\mu\alpha.P', C)$ holds, we have (1) if $\langle \mu\alpha.P', C\rangle \to \langle [\mu\alpha.P'/\alpha]P', C\rangle \xrightarrow{\sigma} \langle Q, C'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$, which is a contradiction. Therefore, $OK([\mu\alpha.P'/\alpha]P', C)$ holds.

- Case $P = P_1; P_2$ and $\dfrac{\langle P_1, C\rangle \overset{\rho}{\Longrightarrow} \langle P_1', C'\rangle}{\langle P_1; P_2, C\rangle \overset{\rho}{\Longrightarrow} \langle P_1'; P_2, C'\rangle}$

  We need to prove $OK_{n'}(P_1'; P_2, C)$, where n' is determined by

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

  Assume that $OK_{n'}(P_1'; P_2, C')$ does not hold. Then, we have (1) $\exists \sigma$, Q and C'' s.t. $\langle P_1'; P_2, C\rangle \xrightarrow{\sigma} \langle Q, C''\rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n'$.

  From the definition of that $OK_n(P_1; P_2, C)$ holds, we have (1) if $\langle P_1; P_2, C\rangle \overset{\rho}{\Longrightarrow} \langle P_1'; P_2, C'\rangle \xrightarrow{\sigma} \langle Q, C''\rangle$, then $\sharp_m(\rho\sigma) - \sharp_f(\rho\sigma) \leq n$.

  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. Therefore, $OK_{n'}(P_1; P_2, F')$ holds.

$\square$

Lemma 3.2. *If consistency(H, R, C) and $\langle H, R, s, n, C\rangle \xrightarrow{\rho} \langle H', R', s', n', C'\rangle$, then consistency$(H', R', C')$.*

Proof. By induction on $\langle H, R, s, n, C\rangle \xrightarrow{\rho} \langle H', R', s', n', C'\rangle$

- Case: $\langle H, R, \textbf{const}(*y)s, n, C\rangle \to \langle H, R, s; \textbf{endconst}(*y), n', C \cup \textbf{const}(*y)\rangle$.

  We need to prove $consistency(H, R, C \cup \textbf{const}(*y))$. From assumption $consistency(H, R, C)$, we have for all x (1) C does not contain both $\textbf{null}(*x)$ and $\neg\textbf{null}(*x)$, therefore $C \cup \textbf{const}(*y)$ does not contain both $\textbf{null}(*x)$ and $\neg\textbf{null}(*x)$. (2) if $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, then $H(R(x)) = null$. Assume that $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, then we have $H(R(x)) = null$. Therefore, $\textbf{const}(*x) \in C \cup \textbf{const}(*y)$ and $\textbf{null}(*x) \in C \cup \textbf{const}(*y)$, then $H(R(x)) = null$. H and R do not change, so $H(R(x)) = null$. Then we get for all x, if $\textbf{const}(*x) \in C \cup \textbf{const}(*y)$ and $\textbf{null}(*x) \in C \cup \textbf{const}(*y)$, then $H(R(x)) = null$. (3) similar to (2).

  Therefore, $consistency(H, R, C \cup \textbf{const}(*y))$ holds.

- Case: $\langle H, R, \textbf{endconst}(*y), n, C\rangle \to \langle H, R, skip, n, C'\rangle$ where $C' = filter(C, *y)$.

  We need to prove $consistency(H, R, C'$ where $C' = filter(C, *y)$. From assumption $consistency(H, R, C)$, we have for all x (1) C does not contain both $\textbf{null}(*x)$ and $\neg\textbf{null}(*x)$. From definition of function $filter(C, *y)$, we know for all x C' does not contain both $\textbf{null}(*x)$ and $\neg\textbf{null}(*x)$. (2) if $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, then $H(R(x)) = null$. Assume that $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, then from the definition of $filter(C, *y)$, we know $\textbf{const}(*x) \in C'$ and $\textbf{null}(*x) \in C'$, and H and R do not change, so $H(R(x)) = null$. Therefore, for all x if $\textbf{const}(*x) \in C'$ and $\textbf{null}(*x) \in C'$, then $H(R(x)) = null$. (3) similar to (2).

  Therefore, $consistency(H, R, C'$ holds.

- Case: $\langle H, R, \textbf{free}(y), n, C\rangle \xrightarrow{\textbf{free}} \langle H \backslash R(y), R, skip, n+1, C\rangle$.

  We need to prove $consistency(H \backslash R(y), R, C$. From assumption $consistency(H, R, C)$, we have for all x (1) C does not contain both $\textbf{null}(*x)$ and $\neg\textbf{null}(*x)$. (2) if $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, then $H(R(x)) = null$. Assume that $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, and we know $(H \backslash R(y))(R(x)) = null$. Therefore, for all x, if $\textbf{const}(*x) \in Cc$ and $\textbf{null}(*x) \in C$, then $(H \backslash R(y))(R(x)) = null$. (3) similar to (2).

  Therefore, $consistency(H \backslash R(y), R, C$ holds.

- Case: $\langle H, R, \textbf{let}\ y = \textbf{mallocin}\ s, n, C\rangle \xrightarrow{\textbf{malloc}} \langle H\{l \mapsto v\}, R\{x' \mapsto l\}, [x'/y]s, n', C\rangle$.

  We need to prove $consistency(H\{l \mapsto v\}, R\{x' \mapsto l\}, C)$. From assumption $consistency(H, R, C)$, we have for all x (1) C does not contain both $\textbf{null}(*x)$ and $\neg\textbf{null}(*x)$. (2) if $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, then $H(R(x)) = null$. Assume that $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, then we have $H(R(x)) = null$, and we know $H\{l \mapsto v\}(Rx' \mapsto v(x)) = null$. Therefore, we get for all x, if $\textbf{const}(*x) \in C$ and $\textbf{null}(*x) \in C$, then $H\{l \mapsto v\}(Rx' \mapsto v(x)) = null$. (3) similar to (2).

  Therefore, $consistency(H\{l \mapsto v\}, R\{x' \mapsto l\}, C)$ holds.

- Case: $\langle H, R, skip; s, n, C\rangle \to \langle H, R, s, n', C\rangle$.

  We need to prove $consistency(H, R, C)$. Obviously, from assumption $consistency(H, R, C)$.

- Case: $\langle H\{R(w) \mapsto v\}, R, *w \leftarrow y, n, C\rangle \to \langle H\{R(w) \mapsto R(y)\}, R, skip, n, C\rangle$ where $\forall z. R(w) = R(z) \Rightarrow \textbf{const}(*z) \notin C$

  We need to prove $consistency(H\{R(w) \mapsto R(y)\}, R, C$. From assumption $consistency(H, R, C)$, we have for all x (1) C does not contain both $\textbf{null}(*x)$ and $\neg\textbf{null}(*x)$ (2) if $\textbf{const}(*x) \in C$ and

$\mathbf{null}(*x) \in C$, then $H\{R(w) \mapsto v\}(R(x)) = null$. Assume that $\mathbf{const}(*x) \in C$ and $\mathbf{null}(*x) \in C$, then we have $H\{R(w) \mapsto v\}(R(x)) = null$, and we know $H\{R(w) \mapsto R(y)\}(R(x)) = null$. Therefore, for all x, if $\mathbf{const}(*x) \in C$ and $\mathbf{null}(*x) \in C$, then $H\{R(w) \mapsto R(y)\}(R(x)) = null$. (3) similar to (2).

Therefore, $consistency(H\{R(w) \mapsto R(y)\}, R, C)$ holds.

- Case: $\langle H, R, \mathbf{let}\ z = y\mathbf{in}\ s, n, C\rangle \rightarrow \langle H, R\{z' \mapsto R(y)\}, [z'/z], n, C\rangle$

  We need to prove $consistency(H, R\{z' \mapsto R(y)\}, C$. From assumption $consistency(H, R, C)$, we have for all x (1) C does not contain both $\mathbf{null}(*x)$ and $\neg\mathbf{null}(*x)$ (2) if $\mathbf{const}(*x) \in C$ and $\mathbf{null}(*x) \in C$, then $H(R(x)) = null$. Assume that $\mathbf{const}(*x) \in C$ and $\mathbf{null}(*x) \in C$, then we have $H(R(x)) = null$, and we get $H(R\{z' \mapsto R(y)\}) = null$. Therefore, for all x, if $\mathbf{const}(*x) \in C$ and $\mathbf{null}(*x) \in C$, then $H(R\{z' \mapsto R(y)\}) = null$. (3) similar to (2).

  Therefore, $consistency(H, R\{z' \mapsto R(y)\}, C)$ holds.

- Case: $\langle H, R, \mathbf{ifnull}\ (*y)\ \mathbf{then}\ s_1\ \mathbf{else}\ \ s_2, n, C\rangle \xrightarrow{\mathbf{null}(*y)} \langle H, R, s_1, n, C\rangle$ where $H(R(y)) = null$ and $\mathbf{const}(*y) \notin C$

  We need to prove $consistency(H, R, C)$. Obviously, $consistency(H, R, C)$ holds from assumption.

- Case: $\langle H, R, \mathbf{ifnull}\ (*y)\ \mathbf{then}\ s_1\ \mathbf{else}\ \ s_2, n, C\rangle \xrightarrow{\neg\mathbf{null}(*y)} \langle H, R, s_2, n, C\rangle$ where $H(R(y)) \neq null$ and $\mathbf{const}(*y) \notin C$

  We need to prove $consistency(H, R, C)$. Obviously, $consistency(H, R, C)$ holds from assumption.

- Case: $\langle H, R, \mathbf{ifnull}\ (*y)\ \mathbf{then}\ s_1\ \mathbf{else}\ \ s_2, n, C\rangle \xrightarrow{\mathbf{null}(*y)} \langle H, R, s_1, n, C \cup \mathbf{null}(*y)\rangle$ where $H(R(y)) = null$ and $\mathbf{const}(*y) \in C$

  We need to prove $consistency(H, R, C \cup \mathbf{null}(*y)$. From assumption $consistency(H, R, C)$, we have for all x (1) C does not contain both $\mathbf{null}(*x)$ and $\neg\mathbf{null}(*x)$. Assume that $\neg\mathbf{null}(*y) \in C$, and because we know $\mathbf{const}(*y) \in C$, then $H(R(y)) \neq null$. Then we get the contradiction $H(R(y))$ should be null. Therefore $\neg\mathbf{null}(*y) \notin C$. Then we get for all x, $C \cup \mathbf{null}(*y)$ does not contain both $\mathbf{null}(*x)$ and $\neg\mathbf{null}(*x)$. (2) if $\mathbf{const}(*x) \in C$ and $\mathbf{null}(*x) \in C$, then $H(R(x)) = null$. Assume that $\mathbf{const}(*x) \in C$ and $\mathbf{null}(*x) \in C$, then we have $H(R(x)) = null$, therefore$\mathbf{const}(*x) \in C \cup \mathbf{null}(*y)$ and $\mathbf{null}(*x) \in C \cup \mathbf{null}(*y)$. H and R do not change, so $H(R(x)) = null$. Therefore, we get for all x, if $\mathbf{const}(*x) \in C \cup \mathbf{null}(*y)$ and $\mathbf{null}(*x) \in C \cup \mathbf{null}(*y)$, then $H(R(x)) = null$. (3) similar to (2).

  Therefore, $consistency(H, R, C \cup *y)$ 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 $\Theta; \Gamma \vdash \langle H, R, \mathbf{const}(*x)s, n, C\rangle : \langle P, C\rangle$, we have $\Theta; \Gamma \vdash \mathbf{const}(*x)s : P$, $OK_n(P, C)$ and $consistency(H, R, C)$. 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 $\langle P, C\rangle \Longrightarrow \langle Q, C \cup \{\mathbf{const}(*x)\}\rangle$ for some Q.

we need to find P' and C' s.t. $\Theta;\Gamma \vdash s;\mathbf{endconst}(*x) : P'$, $OK_n(P',C')$, $\langle P,C'\rangle \Longrightarrow \langle P',C'\rangle$ and $consistency(H,R,C')$. Taking Q as P' and $C \cup \{\mathbf{const}(*x)\}$ as C'. Therefore $\langle P,C\rangle \rightarrow \langle P',C'\rangle$ holds, and then $OK_n(P',C')$ and $consistency(H,R,C')$ hold from Lemma 3.1 and 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 $C' = filter(C,*x)$

  From the assumption $\Theta;\Gamma \vdash \langle H,R,\mathbf{endconst}(*x),n,C\rangle : \langle P,C\rangle$, we have $\Theta;\Gamma \vdash \mathbf{endconst}(*x) : P$, $OK_n(P,C)$ and $consistency(H,R,C)$. 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 , we get $0 \leq Q$ and $\langle P,C\rangle \rightarrow \langle Q,C\rangle$ for some Q.

  we need to find P' and C' s.t. $\Theta;\Gamma \vdash \mathbf{skip} : P'$, $OK_n(P',C')$, $\langle P,C\rangle \Longrightarrow P',C'\rangle$ and $consistency(H,R,C')$. Taking Q as P' and C as C', then $\langle P,C\rangle \rightarrow \langle P',C'\rangle$ holds, and then $OK_n(P',C')$ and $consistency(H,R,C')$ hold from Lemma 3.1 and Lemma 3.2 . 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 $\Theta;\Gamma \vdash \langle H,R,\mathbf{free}(x)\ ,n,C\rangle : \langle P,C\rangle$, we have $OK_n(P,C)$, $consistency(H,R,C)$ 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 $\langle P,F\rangle \xRightarrow{\mathbf{free}} \langle Q,C\rangle$ and $\mathbf{0} \leq Q$ for some Q.

  We need to find P' and C' such that $\langle P,C\rangle \xRightarrow{\mathbf{free}} \langle P',C'\rangle$, $\Theta;\Gamma \vdash \mathbf{skip} : P'$, and $OK_{n+1}(P',C')$. Take Q as P' and C as C'. Then, $\langle P,C\rangle \xRightarrow{\mathbf{free}} \langle P',C'\rangle$ holds, and $OK_{n+1}(P',C')$ holds from Lemma 3.1. We also have $\Theta;\Gamma \vdash \mathbf{skip} : P'$ from T-Skip, $\mathbf{0} \leq Q$ 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 $\Theta;\Gamma \vdash \langle H,R,\mathbf{let}\ x = \mathbf{malloc}()\ \mathbf{in}\ s,n,C\rangle : \langle P,C\rangle$, we have $\Theta;\Gamma \vdash \mathbf{let}\ x = \mathbf{malloc}()\ \mathbf{in}\ s : P$, $OK_n(P,C)$ and $consistency(H,R,C)$. By the inversion of typing rules, we have $\Theta;\Gamma,x \vdash s : P''$ and $\mathbf{malloc};(x)P'' \leq P$ for some P''. By subtyping, we get $\langle P,C\rangle \xRightarrow{\mathbf{malloc}} \langle Q,F\rangle$ and $[x'/x]P'' \leq Q$ for some Q.

  We need to find P' and C' such that $\Theta;\Gamma,x' \vdash [x'/x]s : P'$, $\langle P,C\rangle \xRightarrow{\mathbf{malloc}} \langle P',C'\rangle$, $consistency(H',R',C')$ and $OK_{n-1}(P',C')$. Take Q as P' and C as C'. Then $\langle P,C\rangle \xRightarrow{\mathbf{malloc}} \langle P',C'\rangle$ holds, and then $OK_{n-1}(P',C')$ and $consistency(H',R',C')$ hold by Lemma 3.1 and Lemma 3.2. From $\Theta;\Gamma,x \vdash s : P''$ and $\mathbf{malloc};(x)P'' \leq P$, we have $\Theta;\Gamma,x'' \vdash [x''/x]s : [x''/x]P''$ and $\mathbf{malloc};(x)P'' \leq P$, 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: $\langle H,R,\mathbf{skip};s,n,C\rangle \rightarrow \langle H,R,s,n,C\rangle$

  From the assumption $\Theta;\Gamma \vdash \langle H,R,\mathbf{skip};s,n,C\rangle : \langle P,C\rangle$, we have $\Theta;\Gamma \vdash \mathbf{skip};s : P$, $OK_n(P,C)$ and $consistency(H,R,C)$. 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 $\langle P,C\rangle \Longrightarrow \langle Q,C\rangle$ and $P'' \leq Q$ for some Q.

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

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

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

  We need to find P' and C' such that $\Theta; \Gamma \vdash \mathbf{skip} : P'$, $\langle P, C\rangle \Longrightarrow \langle P', C'\rangle$ and $OK_n(P', C')$. Take P as P' and C as C'. Then $\langle P, C\rangle \Longrightarrow \langle P', C'\rangle$ holds, and then $OK_n(P', C')$ and $consistency(H', R, C')$ hold from Lemma 3.1 and Lemma 3.2. We also have $\Theta; \Gamma \vdash \mathbf{skip} : P'$ from T-SKIP, $0 \leq P$ and T-SUB.

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

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

  We need to find P' and C' such that $\Theta; \Gamma, x', y \vdash [x'/x]s : P'$, $\langle P, C\rangle \rightarrow \langle P', C'\rangle$, $OK_n(P', C')$ and $consistency(H, R', C')$. Take Q as P' and C as C'. Then $\langle P, C\rangle \Longrightarrow \langle P', C'\rangle$ and $OK_n(P', C')$ hold. From $\Theta; \Gamma, x, y \vdash s : P''$ and $\mathbf{let}\ x = y\ \mathbf{in}\ P'' \leq P$, we have $\Theta; \Gamma, x'', y \vdash [x''/x]s : [x''/x]P''$ and $\mathbf{let}\ x'' = y\ \mathbf{in}\ [x''/x]P'' \leq P$, and then by subtying 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: $\langle H, R, \mathbf{let}\ x = \mathbf{null}\ \mathbf{in}\ \ s, n\rangle \rightarrow \langle H, R', [x'/x]s, n\rangle$

  Similar to the above.

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

  Similar to the above.

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

  From assumption $\Theta; \Gamma \vdash \langle H, R, \mathbf{ifnull}\ (*x)\ \mathbf{then}\ s_1\ \mathbf{else}\ \ s_2, n, C\rangle : \langle P, C\rangle$, we have $\Theta; \Gamma \vdash \mathbf{ifnull}\ (*x)\ \mathbf{then}\ s_1\ \mathbf{else}\ \ s_2 : P$, $OK_n(P, C)$ and $consistency(H, R, C)$. 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 $\mathbf{const}(*x) \notin C$, which means $\mathbf{const}(*x) \notin C$, we get $\langle P, C\rangle \xRightarrow{\mathbf{null}(*x)} \langle Q, C\rangle$ and $P_1 \leq Q$ for some Q.

  We need to find P' and C' such that $\Theta; \Gamma \vdash s_1 : P'$, $\langle P, C\rangle \xRightarrow{\mathbf{null}(*x)} \langle P', C'\rangle$ and $OK_n(P', C')$. Take Q as P' and C as C'. Then $\langle P, C\rangle \xRightarrow{\mathbf{null}(*x)} \langle P', C'\rangle$ and $OK_n(P', C')$ 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: $\langle H, R, \mathbf{ifnull}\ (*x)\ \mathbf{then}\ s_1\ \mathbf{else}\ \ s_2, n, C\rangle \xrightarrow{\neg\mathbf{null}(*x)} \langle H, R, s_1, n, C\rangle$ if $H(R(x)) \neq \mathbf{null}$ and $\mathbf{const}(*x) \notin C$

  Similar to the above.

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

  From assumption $\Theta; \Gamma \vdash \langle H, R, \textbf{ifnull } (*x) \textbf{ then } s_1 \textbf{ else } s_2, n, C\rangle : \langle P, C\rangle$, we have $\Theta; \Gamma \vdash \textbf{ifnull } (*x) \textbf{ then } s_1 \textbf{ else } s_2 : P$, $OK_n(P, C)$ and *consistency*(H, R, C). 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 $\textbf{const}(*x) \in C$, we get $\langle P, C\rangle \xRightarrow{\textbf{null}(*x)} \langle Q, C \cup \{\textbf{null}(*x)\}\rangle$ and $P_1 \leq Q$ for some Q.

  We need to find P' and C' such that $\Theta; \Gamma \vdash s_1 : P'$, $\langle P, C\rangle \xRightarrow{\textbf{null}(*x)} \langle P', C'\rangle$, $OK_n(P', C')$ and *consistency*(H, R, C'). Take Q as P' and $C \cup \{\textbf{null}(*x)\}$ as C'. Then $\langle P, C\rangle \xrightarrow{\textbf{null}(*x)} \langle P', c'\rangle$ holds, and then $OK_n(P', C')$ and *consistency*(H, R, C')hold by Lemma 3.1 and Lemma 3.2. 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: $\langle H, R, \textbf{ifnull } (*x) \textbf{ then } s_1 \textbf{ else } s_2, n, C\rangle \xrightarrow{\neg\textbf{null}(*x)} \langle H, R, s_2, n, C'\rangle$ if $H(R(x)) \neq \textbf{null}$, $\textbf{const}(*x) \in C$ and $C' = C \cup \{\neg\textbf{null}(*x)\}$

  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 $\Theta; \Gamma \vdash \langle H, R, s_1; s_2, n, C\rangle : \langle P, C\rangle$, we have $\Theta; \Gamma \vdash s_1; s_2 : P$, $OK_n(P, C)$ and *consistency*(H, R, C). 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 $\exists P_1', C_1'$ s.t. $\Theta; \Gamma \vdash \langle H', R', s_1', n', C'\rangle : \langle P_1', C_1'\rangle$ and $\langle P_1, C\rangle \xrightarrow{\rho} \langle P_1', C_1'\rangle$.

  By subtyping we have $\langle P, C\rangle \xrightarrow{\rho} \langle Q, C_1'\rangle$ and $P_1'; P_2 \leq Q$ for some Q.

  We need to find P' and C' s.t. $\langle P, C\rangle \xrightarrow{\rho} \langle P', C'\rangle$, $OK_n(P', C')$ and $\Theta; \Gamma \vdash s_1'; s_2 : P'\rangle$. Take Q as P' and C_1' as C', $\langle P, C\rangle \xrightarrow{\rho} \langle P', C'\rangle$ and $OK_n(P', C')$ 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* $\Theta \vdash \langle H, R, s, n, C\rangle : \langle P, C\rangle$ *and* $\langle H, R, s, n, C\rangle \xRightarrow{\rho}$ *and* $\rho \in \{\textbf{malloc}, \textbf{free}, \textbf{null}(*x), \neg\textbf{null}(*x)\}$, *then there exists* P' *and* C' *such that* $\langle P, C\rangle \xRightarrow{\rho} \langle P', C'\rangle$.

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

Proof of Lemma 2.3:

By contradiction. Assume $\langle H, R, s, n, C\rangle \xrightarrow{\rho} \textbf{OutOfMemory}$. Then, n is 0 and $\rho = \textbf{malloc}$ from Sem-OutOfMem. From the assumption we have $\Theta; \Gamma \vdash s : P$ and $OK_0(P, C)$. From Lemma 3.3, there exists P' and C' such that $\langle P, C\rangle \xRightarrow{\textbf{malloc}} \langle P', C'\rangle$. However, this contradicts $OK_0(P, C)$.

□

Proof of Theorem 2.1:

We have $\Theta; \emptyset \vdash s : P, \vdash D : \Theta$, $OK_n(P, C)$ and *consistency*(H, R, C).

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

$\square$