A Behavioral Type System for Memory-Leak Freedom

1. abstract

We extend a behavioral type system with dependent types for a program language with manual memory management primitives. The extended type system describes more precise behavior information for a program, which not only can abstract behavior of a program but also deal with path-sensitive statement. By using our current type system with another safe memory deallocation tools, we can even guarantee memory-leak free for nonterminating programs.

2. Introduction

Manual memory mangagement primitives (e.g. `malloc` and `free` in C language) are a very flexible way to manage computer memory cells. We can write a program which dynamically allocates a memory cell during running and deallocates a memory cell when it is no longer used. However, manual memory management primitives often cause hard-to-find problems, for example, double frees (`free` a deallocated memory cell), memory leaks (forget to deallocate memory cells) and illegal accesses to a dangling pointer. Therefore, many static verification methods have been proposed to guarantee safe memory deallocation. They prove *partial* memory-leak freedom: if a program terminates, all the memory cells are safe deallocated. As we know that nonterminating programs are very common in real-world programmings such as Web servers and operating systems. To guarantee *total* memory-leak freedom, if a program does not consume unbounded number of memory cells during execution, is a very crucial issue.

3. Language $\mathscr{L}$

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

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

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

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 $\mathbf{Dom}(f)$ 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:

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

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

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

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 $*x \leftarrow y$ 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 $\mathbf{let}\ x = e\ \mathbf{in}\ s$ evaluates the expression e, binds x to the result, and executes s. The expression $\mathbf{malloc}()$ allocates a new memory cell. The expression **null** evaluates to the null pointer. The expression $*y$ means dereferencing a memory cell pointed to by y. The statement $\mathbf{ifnull}\ (*x)\mathbf{then}\ s_1\mathbf{else}\ s_2$ 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 $\mathbf{const}(*x)s$ 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 $\langle d_1 \cup \ldots d_n \rangle$, 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.

3.1. Operational semantics

In this section we introduce operational semantics of language $\mathscr{L}$. We assume there is a countable infinite set $\mathscr{H}$ 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 $\mathscr{H}$ to $\mathscr{H} \cup \{\mathbf{null}\}$;
- R, an *environment*, is a finite mapping from **Var** to $\mathscr{H} \cup \{\mathbf{null}\}$;

- 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 $\mathbf{const}(*x)$, $\mathbf{null}(*x)$ and $\neg\mathbf{null}(*x)$.

The operational semantics of the language $\mathscr{L}$ 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.

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

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 $\rho_1 \ldots \rho_n$. We write $\xrightarrow{\rho_1\cdots\rho_n}_D$ for $\xrightarrow{\rho_1}_D\xrightarrow{\rho_2}_D \ldots \xrightarrow{\rho_n}_D$. We write $\overset{\rho}{\Longrightarrow}_D$ for $\longrightarrow^*_D\xrightarrow{\rho}_D\longrightarrow^*_D$. We write $\overset{\rho_1\cdots\rho_n}{\Longrightarrow}_D$ for $\overset{\rho_1}{\Longrightarrow}_D \ldots \overset{\rho_n}{\Longrightarrow}_D$.

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 stamtement s is expressed by adding a statement $\mathbf{endconst}(*x)$ 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 $\mathbf{free}(x)$, 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 $\mathscr{L}$, 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 $\langle H,R,s,n,C\rangle \overset{\sigma}{\Longrightarrow}$* **OutOfMemory***. A program $\langle D,s\rangle$* consumes at least n cells *if $\langle \emptyset,\emptyset,s,n,\emptyset\rangle$ goes overflow.*

4. Type system

4.1. Types

The syntax of the types is as follows.

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

Behavioral types ranged over by P express the abstaction 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.

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.

$$\frac{C' = filter\_C(C,*x)}{\langle H,R,\mathbf{endconst}(*x),n,C\rangle \to_D \langle H,R,\mathbf{skip},n,C'\rangle} \quad \text{(SEM-CONSTSKIP)}$$

$$\langle H,R,\mathbf{const}(*x)s,n,C\rangle \to_D \langle H,R,s;\mathbf{endconst}(*x),n,C\cup\{\mathbf{const}(*x)\}\rangle \quad \text{(SEM-CONSTSEQ)}$$

$$\langle H,R,\mathbf{skip};s,n,C\rangle \longrightarrow_D \langle H,R,s,n,C\rangle \quad \text{(SEM-SKIP)}$$

$$\frac{\langle H,R,s_1,n,C\rangle \xrightarrow{\rho}_D \langle H',R',s_1',n',C'\rangle}{\langle H,R,s_1;s_2,n,C\rangle \xrightarrow{\rho}_D \langle H',R',s_1';s_2,n',C'\rangle} \quad \text{(SEM-SEQ)}$$

$$\frac{x' \notin \mathbf{Dom}(R)}{\langle H,\ R,\ \mathbf{let}\ x=\mathbf{null}\ \mathbf{in}\ s,n,C\rangle \longrightarrow_D \langle H,\ R\{x'\mapsto \mathbf{null}\},\ [x'/x]s,n,C\rangle} \quad \text{(SEM-LETNULL)}$$

$$\frac{x' \notin \mathbf{Dom}(R)}{\langle H,\ R,\ \mathbf{let}\ x=y\ \mathbf{in}\ s,n,C\rangle \longrightarrow_D \langle H,\ R\{x'\mapsto R(y)\},\ [x'/x]s,n,C\rangle} \quad \text{(SEM-LETEQ)}$$

$$\frac{H(R(x))=\mathbf{null}, \mathbf{const}(*x)\notin C}{\langle H,\ R,\ \mathbf{ifnull}\ (*x)\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2,\ n,C\rangle \to_D \langle H,\ R,\ s_1,\ n,C\rangle} \quad \text{(SEM-IFNULLT)}$$

$$\frac{H(R(x))\neq\mathbf{null}, \mathbf{const}(*x)\notin C}{\langle H,\ R,\ \mathbf{ifnull}\ (*x)\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2,\ n,C\rangle \to_D \langle H,\ R,\ s_2,\ n,C\rangle} \quad \text{(SEM-IFNULLF)}$$

$$\frac{H(R(x))=\mathbf{null}, \mathbf{const}(*x)\in C}{\langle H,\ R,\ \mathbf{ifnull}\ (*x)\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2,\ n,C\rangle \to_D \langle H,\ R,\ s_1,\ n,C\cup\{\mathbf{null}(*x)\}\rangle} \quad \text{(SEM-IFCONSTNULLT)}$$

$$\frac{H(R(x))\neq\mathbf{null}, \mathbf{const}(*x)\in C}{\langle H,\ R,\ \mathbf{ifnull}\ (*x)\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2,\ n,C\rangle \to_D \langle H,\ R,\ s_2,\ n,C\cup\{\neg\mathbf{null}(*x)\}\rangle} \quad \text{(SEM-IFCONSTNULLF)}$$

$$\frac{\mathbf{const}(*x)\notin C}{\langle H\{R(x)\mapsto v\},R,*x\leftarrow y,n,C\rangle \longrightarrow_D \langle H\{R(x)\mapsto R(y)\},R,\mathbf{skip},n,C\rangle} \quad \text{(SEM-ASSIGN)}$$

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

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

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

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

$$\frac{R(x)=\mathbf{null}\ \text{or}\ R(x)\notin\mathbf{Dom}(H)}{\langle H,\ R,\ \mathbf{free}(x),n,C\rangle \xrightarrow{\mathbf{free}}_D \mathbf{MemEx}} \quad \text{(SEM-FREEEXN)}$$

$$\frac{R(x)=\mathbf{null}\ \text{or}\ R(x)\notin\mathbf{Dom}(H)}{\langle H,\ R,\ *x\leftarrow y,n,C\rangle \longrightarrow_D \mathbf{MemEx}} \quad \text{(SEM-ASSIGNEXN)} \qquad \frac{R(y)=\mathbf{null}\ \text{or}\ R(y)\notin\mathbf{Dom}(H)}{\langle H,\ R,\ \mathbf{let}\ x=*y\ \mathbf{in}\ s,n,C\rangle \longrightarrow_D \mathbf{MemEx}} \quad \text{(SEM-DEREFEXN)}$$

$$\frac{\forall z.\mathbf{const}(*z)\in C\ \text{and}\ R(x)=R(z)}{\langle H\{R(x)\mapsto v\},R,*x\leftarrow y,n,C\rangle \longrightarrow_D \mathbf{ConstEx}} \quad \text{(SEM-ASSIGNCONSTEXN)}$$

$$\langle H,\ R,\ \mathbf{let}\ x=\mathbf{malloc}()\ \mathbf{in}\ s,0,C\rangle \xrightarrow{\mathbf{malloc}(x')}_D \mathbf{OutOfMemory} \quad \text{(SEM-OUTOFMEM)}$$

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

$$\langle \mathbf{0};P,F\rangle \to \langle P,F\rangle$$

$$\langle \mathbf{free},F\rangle \xrightarrow{\mathbf{free}} \langle \mathbf{0},F\rangle \quad \text{(TR-FREE)}$$

$$\langle P_1+P_2,F\rangle \to \langle P_1,F\rangle \quad \text{(TR-CHOICEL)}$$

$$\frac{\langle P_1,F\rangle \xrightarrow{\rho} \langle P_1',F'\rangle}{\langle P_1;P_2,F\rangle \xrightarrow{\rho} \langle P_1';P_2,F}$$

$$\langle \mathbf{let}\ x=\mathbf{malloc\ in}\ P,F\rangle \xrightarrow{\mathbf{malloc}(x')}$$

$$\langle \mathbf{let}\ x=y\ \mathbf{in}\ P,F\rangle \to \langle [x'/x]$$

$$\langle \mathbf{let}\ x=*y\ \mathbf{in}\ P,F\rangle \to \langle [x'/x$$

$$\langle \mathbf{let}\ x=\mathbf{null\ in}\ P,F\rangle \to \langle [x'/$$

$$\langle \mathbf{const}(*x)P,F\rangle \to \langle P;\mathbf{endconst}(*x),F$$

$$\frac{F' = filter\_T(F,*x)}{\langle \mathbf{endconst}(*x),F\rangle \to \langle \mathbf{0},}$$

$$\frac{\mathbf{const}(*x)\notin F}{\langle (*x)(P_1,P_2),F\rangle \to \langle P_1,F\rangle} \quad \text{(TR-NOTCONST1)}$$

$$\frac{\mathbf{null}(*x)\in F \qquad \mathbf{const}(*x)\in F}{\langle (*x)(P_1,P_2),F\rangle \to \langle P_1,F\rangle} \quad \text{(TR-NULLIN)}$$

$$\frac{\mathbf{null}(*x),\neg\mathbf{null}(*x)\notin F \qquad \mathbf{cons}}{\langle (*x)(P_1,P_2),F\rangle \to \langle P_1,F\cup\mathbf{nu}}$$

$$\frac{\mathbf{null}(*x),\neg\mathbf{null}(*x)\notin F \qquad \mathbf{cons}}{\langle (*x)(P_1,P_2),F\rangle \to \langle P_2,F\cup\neg\mathbf{n}}$$

Figure 2: semantics of behavioral types with dependent types.

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

$filter\_T(F,*x)$ = *let* $F' = F$ *in*
if $\mathbf{const}(*x)\notin F'$ *then return* $(F'\backslash\{\mathbf{null}(*x),\neg\mathbf{nul}$
else return F'

4.2. 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 3: typing rules

Definition 3 ($\sharp_\rho(\sigma)$)**.** *$\sharp_\rho(\sigma)$ 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 $\sharp_m(\sigma) - \sharp_f(\sigma) \leq n$ 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 $\langle P_2,F\rangle \stackrel{\rho}{\Longrightarrow} \langle P_2',F'\rangle$ 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.*

4.3. Type soundness

Theorem 4.1. *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 4.2 (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) $\langle P,F\rangle \stackrel{\rho}{\Longrightarrow} \langle P',F'\rangle$.*

Lemma 4.3 (Lack of immediate overflow)**.** *If $\Theta;\Gamma \vdash \langle H,R,s,n,C\rangle : \langle P,F\rangle$, then $\langle H,R,s,n,C\rangle \not\xrightarrow{\mathbf{malloc}}$* **OutOfMemory***.*

5. Experiments

6. Related Works

[todo]

7. Conclusion

[todo]

8. Acknowledgements

Appendix

9. Proof of Lemmas

Lemma 9.1. *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 $\langle \mathbf{0};P',F\rangle \to \langle P',F\rangle$
  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 $P = \mathbf{let}\ x = \mathbf{malloc}\ \mathbf{in}\ P'$ and $\langle \mathbf{let}\ x = \mathbf{malloc}\ \mathbf{in}\ P',F\rangle \xrightarrow{\mathbf{malloc}(x')} \langle [x'/x]P',F\rangle$
  Similiar to above.
- Case $P = \mathbf{let}\ x = y\ \mathbf{in}\ P'$ and $\langle \mathbf{let}\ x = y\ \mathbf{in}\ P',F\rangle \to \langle [x'/x]P',F\rangle$
  Similiar to above.
- Case $P = \mathbf{let}\ x = *y\ \mathbf{in}\ P'$ and $\langle \mathbf{let}\ x = *y\ \mathbf{in}\ P',F\rangle \to \langle [x'/x]P',F\rangle$
  Similiar to above.
- Case $P = \mathbf{let}\ x = \mathbf{null}\ \mathbf{in}\ P'$ and $\langle \mathbf{let}\ x = \mathbf{null}\ \mathbf{in}\ P',F\rangle \to \langle [x'/x]P',F\rangle$
  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 $\frac{\mathbf{const}(*x)\notin F}{\langle (*x)(P_1,P_2),F\rangle \to \langle P_1,F\rangle}$
  We need to prove $OK(F)$. From the assumption, $OK(F)$ holds.
- Case $P = (*x)(P_1,P_2)$ and $\frac{\mathbf{const}(*x)\notin F}{\langle (*x)(P_1,P_2),F\rangle \to \langle P_2,F\rangle}$
  We need to prove $OK(F)$. From the assumption, $OK(F)$ holds.
- Case $P = (*x)(P_1,P_2)$ and $\frac{\mathbf{null}(*x)\in F \qquad \mathbf{const}(*x)\in F}{\langle (*x)(P_1,P_2),F\rangle \to \langle P_1,F\rangle}$
  We need to prove $OK(F)$. From the assumption, $OK(F)$ holds.

- Case $P = (*x)(P_1, P_2)$ and $\frac{\neg\mathbf{null}(*x) \in F \qquad \mathbf{const}(*x) \in F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_2, F\rangle}$
We need to prove $OK(F)$. From the assumption, it holds.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin F \qquad \mathbf{const}(*x) \in F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_1, F \cup \mathbf{null}(*x)\rangle}$
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 $\frac{\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin F \qquad \mathbf{const}(*x) \in F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_2, F \cup \neg\mathbf{null}(*x)\rangle}$
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 $\langle \mathbf{const}(*x)P', F\rangle \to \langle P'; \mathbf{endconst}(*x), F \cup \{\mathbf{const}(*x)\}\rangle$
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 $\frac{F' = filter\_T(F, *x)}{\langle \mathbf{endconst}(*x), F\rangle \to \langle \mathbf{0}, F'\rangle}$
we need to prove $OK(F')$. Form 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 $F' = F \backslash \{\mathbf{null}(*x), \neg\mathbf{null}(*x)\}$ 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 $\langle \mu\alpha.P', F\rangle \to \langle [\mu\alpha.P']P', F\rangle$
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 9.2. *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 $\rho = Otherwise$*

Proof. By induction on $\langle P, F\rangle \xrightarrow{\rho} \langle P', F'\rangle$.
- Case $P = \mathbf{0}; P'$ and $\langle \mathbf{0}; P', F\rangle \to \langle P', F\rangle$
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 $\langle \mathbf{0}; P', F\rangle \to \langle P', F\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ 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 $\sharp_m(\sigma) - \sharp_f(\sigma) \le n-1$ 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 $\langle \mathbf{let}\ x = y\ \mathbf{in}\ P', F\rangle \to \langle [x'/x]P', F\rangle$
Similar to the above.
- Case $P = \mathbf{let}\ x = *y\ \mathbf{in}\ P'$ and $\langle \mathbf{let}\ x = *y\ \mathbf{in}\ P', F\rangle \to \langle [x'/x]P', F\rangle$
Similar to the above.
- Case $P = \mathbf{let}\ x = \mathbf{null}\ \mathbf{in}\ P'$ and $\langle \mathbf{let}\ x = \mathbf{null}\ \mathbf{in}\ P', F\rangle \to \langle [x'/x]P', F\rangle$
Similar to the above.
- Case $P = \mathbf{free}$ and $\langle \mathbf{free}, F\rangle \xrightarrow{\mathbf{free}} \langle \mathbf{0}, F\rangle$
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 $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ 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 9.1. Therefore, $OK(\mathbf{0}, F)$ holds.
- Case $P = \mathbf{endconst}(*x)$ and $\frac{F' = filter\_T(F, *x)}{\langle \mathbf{endconst}(*x), F\rangle \to \langle \mathbf{0}, F'\rangle}$
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 $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ 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 $\mathbf{null}(*x)$ and $\neg\mathbf{null}(*x)$. By the definition of function $filter\_T$, we have $F' = F \backslash \{\mathbf{null}(*x), \neg\mathbf{null}(*x)\}$ or $F - \mathbf{const}(*x)$. Therefore $OK(F')$ holds. So $OK_n(\mathbf{0}, F')$ holds.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{const}(*x) \notin F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_1, F\rangle}$
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 $\langle (*x)(P_1, P_2), F\rangle \to \langle P_1, F\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$ then $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ 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 $\frac{\mathbf{const}(*x) \notin F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_2, F\rangle}$
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 $\langle (*x)(P_1, P_2), F\rangle \to \langle P_2, F\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ 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 $\frac{\mathbf{null}(*x) \in F \qquad \mathbf{const}(*x) \in F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_1, F\rangle}$
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 $\langle (*x)(P_1, P_2), F\rangle \to \langle P_1, F\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$, then

$\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ 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 $\frac{\neg\mathbf{null}(*x) \in F \qquad \mathbf{const}(*x) \in F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_2, F\rangle}$
  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 $\langle (*x)(P_1, P_2), F\rangle \to \langle P_2, F\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ 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 $\frac{\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin F \qquad \mathbf{const}(*x) \in F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_1, F \cup \{\mathbf{null}(*x)\}\rangle}$
  We need to prove $OK_n(P_1, F \cup \{\mathbf{null}(*x)\})$. Assume that $OK_n(P_1, F \cup \{\mathbf{null}(*x)\})$ does not hold. Then we have (1) $\exists\sigma$ and Q s.t. $\langle P_1, F \cup \{\mathbf{null}(*x)\}\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F \cup \{\mathbf{null}(*x)\})$ does not hold.
  From the definition of that $OK_n((*x)(P_1, P_2), F)$ holds, we have (1) if $\langle (*x)(P_1, P_2), F\rangle \to \langle P_1, F \cup \{\mathbf{null}(*x)\}\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ and (2) $OK(F)$ holds. By $OK(F)$ and $\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin F$, we have $OK(F \cup \{\mathbf{null}(*x)\})$ holds. Therefore, we get the contradiction and $OK_n(P_1, F \cup \{\mathbf{null}(*x)\})$ holds.
- Case $P = (*x)(P_1, P_2)$ and $\frac{\mathbf{null}(*x), \neg\mathbf{null}(*x) \notin F \qquad \mathbf{const}(*x) \in F}{\langle (*x)(P_1,P_2), F\rangle \to \langle P_2, F \cup \{\neg\mathbf{null}(*x)\}\rangle}$
  Similar to the above.
- Case $P = \mathbf{const}(*x)P'$ and $\langle \mathbf{const}(*x)P', F\rangle \to \langle P'; \mathbf{endconst}(*x), F \cup \mathbf{const}(*x)\rangle$
  We need to prove $OK_n(P'; \mathbf{endconst}(*x), F \cup \mathbf{const}(*x))$. Assume that $OK_n(P'; \mathbf{endconst}(*x), F \cup \mathbf{const}(*x))$ does not hold. Then, we have (1) $\exists\sigma$ and Q s.t. $\langle P'; \mathbf{endconst}(*x), F \cup \mathbf{const}(*x)\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$ and $\sharp_m(\sigma) - \sharp_f(\sigma) > n$ or (2) $OK(F \cup \mathbf{const}(*x))$ does not hold.
  From the definition of that $OK_n(\mathbf{const}(*x)P', F)$ holds, we have (1) if $\langle \mathbf{const}(*x)P', F\rangle \to \langle P; \mathbf{endconst}(*x), F \cup \mathbf{const}(*x)\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$ 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 $\langle \mu\alpha.P', F\rangle \to \langle [\mu\alpha.P'/\alpha]P', F\rangle$
  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 $\langle \mu\alpha.P', F\rangle \to \langle [\mu\alpha.P'/\alpha]P', F\rangle \xrightarrow{\sigma} \langle Q, F'\rangle$, then $\sharp_m(\sigma) - \sharp_f(\sigma) \le n$, which is a contradiction; and (2) $OK(F)$ holds. From the Lemma 9.1, $OK(F \cup \neg\mathbf{null}(*x))$ holds. Therefore, $OK([\mu\alpha.P'/\alpha]P', F)$ holds.
- Case $P = P_1; P_2$ and $\frac{\langle P_1, F\rangle \overset{\rho}{\Longrightarrow} \langle P_1', F'\rangle}{\langle P_1; P_2, F\rangle \overset{\rho}{\Longrightarrow} \langle P_1'; P_2, F'\rangle}$
  We need to prove $OK_{n'}(P_1'; P_2, F)$, where n' is determined by
  $$n' = \begin{cases} n+1 & \rho = \mathbf{free} \\ n-1 & \rho = \mathbf{malloc} \\ n & \text{Otherwise.} \end{cases}$$
  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 $\langle P_1; P_2, F\rangle \overset{\rho}{\Longrightarrow} \langle P_1'; P_2, F'\rangle \xrightarrow{\sigma} \langle Q, F''\rangle$, then $\sharp_m(\rho\sigma) - \sharp_f(\rho\sigma) \le n$ and (2) $OK(F)$ holds.
  From (1), we get $n' + \sharp_m(\rho) - \sharp_f(\rho) < \sharp_m(\rho) + \sharp_m(\sigma) - \sharp_f(\rho) - \sharp_f(\sigma) \le n$. 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.

$\square$

Proof of Lemma 4.2: By induction on the derivation of $\langle H, R, s, n, C\rangle \xrightarrow{\rho} \langle H', R', s', n', C'\rangle$.

- Case: $\langle H, R, \mathbf{const}(*x)s, n, C\rangle \to \langle H, R, s; \mathbf{endconst}(*x), n, C \cup \{\mathbf{const}(*x)\}\rangle$
  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 $\mathbf{const}(*x)P'' \le P$ for some P''. By subtyping, we have $P''; \mathbf{endconst}(*x) \le Q$ and $\langle P, F\rangle \Longrightarrow \langle Q, F \cup \{\mathbf{const}(*x)\}\rangle$ 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 $\langle P, F'\rangle \Longrightarrow \langle P', F'\rangle$. Taking Q as P' and $F \cup \{\mathbf{const}(*x)\}$ as F'. Therefore $\langle P, F\rangle \to \langle P', F'\rangle$ holds, and $OK_n(P', F')$ holds from Lemma 9.2. From $\Theta; \Gamma \vdash s; \mathbf{endconst}(*x) : P''; \mathbf{endconst}(*x)$, $P''; \mathbf{endconst}(*x) \le Q$ and T-SUB, $\Theta; \Gamma \vdash s; \mathbf{endconst}(*x) : P'$ holds.
- Case: $\langle H, R, \mathbf{endconst}(*x), n, C\rangle \to \langle H, R, \mathbf{skip}, n, C'\rangle$ where $C' = filter\_C(C, *x)$
  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 $\mathbf{endconst}(*x) \le P$. By subtyping and function $filter\_T(F, *x)$, we get $0 \le Q$ and $\langle P, F\rangle \to \langle Q, F''\rangle$ for some Q.
  we need to find P' and F' s.t. $\Theta; \Gamma \vdash \mathbf{skip} : P'$, $OK_n(P', F')$ and $\langle P, F\rangle \Longrightarrow P', F'\rangle$. Taking Q as P' and F'' as F' therefore $F' \approx C'$ from functions $filter\_T(F, *x)$ and $filter\_C(C, *x)$; $\langle P, F\rangle \to \langle P', F'\rangle$ and $OK_n(P', F')$ hold. From T-SKIP, T-SUB and $0 \le Q$, 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 $\mathbf{free} \le P$. By the subtyping, we have $\langle P, F\rangle \overset{\mathbf{free}}{\Longrightarrow} \langle Q, F\rangle$ and $\mathbf{0} \le Q$for some Q.
  We need to find P' and F' such that $\langle P, F\rangle \overset{\mathbf{free}}{\Longrightarrow} \langle P', F'\rangle$, $\Theta; \Gamma \vdash \mathbf{skip} : P'$, and $OK_{n+1}(P', F')$. Take Q as P' and F as F'. Then, $\langle P, F\rangle \overset{\mathbf{free}}{\Longrightarrow} \langle P', F'\rangle$ holds, and $OK_{n+1}(P', F')$ holds from Lemma 9.2. We also have $\Theta; \Gamma \vdash \mathbf{skip} : P'$ from

T-SKIP, $\mathbf{0} \leq Q$ and T-SUB.

- Case: $\langle H,R,\textbf{let}\ x = \textbf{malloc}()\ \textbf{in}\ s,n,C\rangle \xrightarrow{\textbf{malloc}} \langle H',R',[x'/x]s,n-1,C\rangle$
From the assumption $\Theta;\Gamma \vdash \langle H,R,\textbf{let}\ x = \textbf{malloc}()\ \textbf{in}\ s,n,C\rangle : \langle P,F\rangle$, we have $\Theta;\Gamma \vdash \textbf{let}\ x = \textbf{malloc}()\ \textbf{in}\ s : P$ and $OK_n(P,F)$. By the inversion of typing rules, we have $\Theta;\Gamma,x \vdash s : P''$ and $\textbf{let}\ x = \textbf{malloc in}\ P'' \leq P$ for some P''. By subtyping, we get $\langle P,F\rangle \xRightarrow{\textbf{malloc}x'} \langle Q,F\rangle$ 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 $\langle P,F\rangle \xRightarrow{\textbf{malloc}x'} \langle P',F'\rangle$ and $OK_{n-1}(P',F')$. Take Q as P' and F as F'. Then $\langle P,F\rangle \xRightarrow{\textbf{malloc}x'} \langle P',F'\rangle$ holds, and $OK_{n-1}(P',F')$ holds by Lemma 9.2. From $\Theta;\Gamma,x \vdash s : P''$ and $\textbf{let}\ x = \textbf{malloc in}\ P'' \leq P$, we have $\Theta;\Gamma,x'' \vdash [x''/x]s : [x''/x]P''$ and $\textbf{let}\ x'' = \textbf{malloc in}\ [x''/x]P'' \leq P$, 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: $\langle H,R,\textbf{skip};s,n,C\rangle \to \langle H,R,s,n,C\rangle$
From the assumption $\Theta;\Gamma \vdash \langle H,R,\textbf{skip};s,n,C\rangle : \langle P,F\rangle$, we have $\Theta;\Gamma \vdash \textbf{skip};s : P$ 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 $\langle P,F\rangle \Longrightarrow \langle Q,F\rangle$ and $P'' \leq Q$ for some Q.
We need to find P' and F' such that $\Theta;\Gamma \vdash s : P'$ and $\langle P,F\rangle \to \langle P',F'\rangle$ and $OK_n(P',F')$. Take Q as P' and F as F'. Then $\langle P,F\rangle \Longrightarrow \langle P',F'\rangle$ 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: $\langle H,R,*x \leftarrow y,n,C\rangle \to \langle H',R,\textbf{skip},n,C\rangle$
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 $\Theta;\Gamma \vdash \textbf{skip} : P'$, $\langle P,F\rangle \Longrightarrow \langle P',F'\rangle$ and $OK_n(P',F')$. Take P as P' and F as F'. Then, $\langle P,F\rangle \Longrightarrow \langle P',F'\rangle$ and $OK_n(P',F')$ hold. We also have $\Theta;\Gamma \vdash \textbf{skip} : P'$ from T-SKIP, $0 \leq P$ and T-SUB.
- Case: $\langle H,R,\textbf{let}\ x = y\ \textbf{in}\ s,n,C\rangle \to \langle H,R',[x'/x]s,n,C\rangle$
From the assumption $\Theta;\Gamma \vdash \langle H,R,\textbf{let}\ x = y\ \textbf{in}\ s,n,C\rangle : \langle P,F\rangle$, we have $\Theta;\Gamma,y \vdash \textbf{let}\ x = y\ \textbf{in}\ s : P$ and $OK_n(P,F)$. From the inversion of typing rules, we have $\Theta;\Gamma,x,y \vdash s : P''$ and $\textbf{let}\ x = y\ \textbf{in}\ P'' \leq P$ for some P''. By subtying, we have $\langle P,F\rangle \to \langle Q,F\rangle$ and $[x'/x]P'' \leq Q$ for some Q.
We need to find P' and F' such that $\Theta;\Gamma,x',y \vdash [x'/x]s : P'$, $\langle P,F\rangle \to \langle P',F'\rangle$ and $OK_n(P',F')$. Take Q as P' and F as F'. Then $\langle P,F\rangle \Longrightarrow \langle P',F'\rangle$ and $OK_n(P',F')$ hold. From $\Theta;\Gamma,x,y \vdash s : P''$ and $\textbf{let}\ x = y\ \textbf{in}\ P'' \leq P$, we have $\Theta;\Gamma,x'',y \vdash [x''/x]s : [x''/x]P''$ and $\textbf{let}\ x'' = y\ \textbf{in}\ [x''/x]P'' \leq P$, 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: $\langle H,R,\textbf{let}\ x = \textbf{null in}\ s,n\rangle \to \langle H,R',[x'/x]s,n\rangle$
Similar to the above.
- Case: $\langle H,R,\textbf{let}\ x = *y\ \textbf{in}\ s,n\rangle \to \langle H,R',[x'/x]s,n\rangle$
Similar to the above.
- Case: $\langle H,R,\textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,n,C\rangle \to \langle H,R,s_1,n,C\rangle$ if $H(R(x)) = \textbf{null}$ and $\textbf{const}(*x) \notin C$
From assumption $\Theta;\Gamma \vdash \langle H,R,\textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,n,C\rangle : \langle P,F\rangle$, we have $\Theta;\Gamma \vdash \textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2 : P$ 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 $\textbf{const}(*x) \notin C$, which means $\textbf{const}(*x) \notin F$, we get $\langle P,F\rangle \Longrightarrow \langle Q,F\rangle$ 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 $\langle P,F\rangle \to \langle P',F'\rangle$ 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: $\langle H,R,\textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,n,C\rangle \to \langle H,R,s_1,n,C\rangle$ if $H(R(x)) \neq \textbf{null}$ and $\textbf{const}(*x) \notin C$
Similar to the above.
- Case: $\langle H,R,\textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,n,C\rangle \to \langle H,R,s_1,n,C'\rangle$ if $H(R(x)) = \textbf{null}$, $\textbf{const}(*x) \in C$ and $C' = C \cup \{\textbf{null}(*x)\}$
From assumption $\Theta;\Gamma \vdash \langle H,R,\textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,n,C\rangle : \langle P,F\rangle$, we have $\Theta;\Gamma \vdash \textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2 : P$ 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, $\textbf{const}(*x) \in C$ and $\textbf{assume}(*x \neq \textbf{null}) \notin C$ which are similar to $\textbf{const}(*x) \in F$ and $\neg\textbf{null}(*x) \notin F$, we get $\langle P,F\rangle \Longrightarrow \langle Q,F \cup \{\textbf{null}(*x)\}\rangle$ 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 $F \cup \{\textbf{null}(*x)\}$ as F'. Then $C' \approx F'$, $\langle P,F\rangle \to \langle P',F'\rangle$ 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: $\langle H,R,\textbf{ifnull}\ (*x)\ \textbf{then}\ s_1\ \textbf{else}\ s_2,n,C\rangle \to \langle H,R,s_2,n,C'\rangle$ 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: $\langle H,R,s_1;s_2,n,C\rangle \to \langle H',R',s_1';s_2,n',C'\rangle$
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 $\Theta;\Gamma \vdash s_1';s_2 : P'\rangle$. 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.

$\square$

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

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

Proof of Lemma 4.3:

By contradiction. Assume $\langle H,R,s,n,C\rangle \xrightarrow{\rho}$ **OutOfMemory**. Then, n is 0 and $\rho = \textbf{malloc}$ from SEM-OUTOFMEM. From the assumption we have $\Theta;\Gamma \vdash s : P$ and $OK_0(P,F)$. From Lemma 9.3, there exists P' and F' such that $\langle P,F\rangle \overset{\textbf{malloc}}{\Longrightarrow} \langle P',F'\rangle$. However, this contradicts $OK_0(P,F)$.

□

Proof of Theorem 4.1:

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

Suppose that there exists σ such that $\langle \emptyset,\emptyset,s,n,C\rangle \xrightarrow{\sigma} \langle H',R',s',n',C'\rangle \xrightarrow{\rho}$ **OutOfMemory**. Then, $n' = 0$ and $\rho = \textbf{malloc}$. From Lemma 4.2, there exists P' and F' such that $\Theta;\Gamma' \vdash s' : P'$, $\langle P,F\rangle \overset{\sigma}{\Longrightarrow} \langle P',F'\rangle$, and $OK_0(P',F')$; hence $\langle H',R',s',0\rangle \xrightarrow{\textbf{malloc}}$. However, this contradicts Lemma 4.3.

□

10. Syntax Directed Typing Rules

$$\frac{C = \emptyset}{\Theta;\Gamma;C \vdash \textbf{skip} : \textbf{0}} \quad \text{(ST-SK}$$

$$\frac{\Theta;\Gamma;C_1 \vdash s_1 : P_1 \quad \Theta;\Gamma;C_2 \vdash s_2 : P_2 \quad C = C_1 \cup C_2}{\Theta;\Gamma;C \vdash s_1;s_2 : P}$$

$$\frac{\Theta;\Gamma;C_1 \vdash y \quad \Theta;\Gamma;C_2 \vdash x : \quad C = C_1 \cup}{\Theta;\Gamma;C \vdash *x \leftarrow y : \textbf{0}}$$

$$\frac{\Theta;\Gamma;C_1 \vdash x \qquad C = C_1}{\Gamma;C \vdash \textbf{free}(x) : \textbf{free}} \quad \text{(ST}$$

$$\frac{\Theta;\Gamma,x;C_1 \vdash s : P_1 \quad C = C_1 \cup \{P_1 \leq P}{\Theta;\Gamma;C \vdash \textbf{let}\ x = \textbf{malloc}()\ \textbf{in}\ s : \textbf{malloc}}$$

$$\frac{\Theta;\Gamma;C_1 \vdash y \quad \Theta;\Gamma,x;C_2 \vdash s : P_1 \quad C = C_1 \cup C_2 \cup}{\Theta;\Gamma;C \vdash \textbf{let}\ x = y\ \textbf{in}\ s : P}$$

$$\frac{\Theta;\Gamma;C_1 \vdash y \quad \Theta;\Gamma,x;C_2 \vdash s : P_1 \quad C = C_1 \cup C_2 \cup}{\Theta;\Gamma;C \vdash \textbf{let}\ x = *y\ \textbf{in}\ s : P}$$

$$\frac{\Theta;\Gamma;C_1 \vdash x \quad \Theta;\Gamma;C_2 \vdash s_1 : P_1 \quad \Theta;\Gamma;C_3 \vdash s_2 : P_2 \quad C = C_1 \cup C}{\Theta;\Gamma;C \vdash \textbf{ifnull}\ (*x) \textbf{then}\ s_1 \textbf{else}\ s_2 :}$$

$$\frac{\Theta(f) = P_1 \quad C = P_1 \leq P}{\Gamma,\vec{x} : \vec{\tau} \vdash f(\vec{x}) : P} \quad \text{(S}$$

$$\frac{\Theta \vdash D : \Theta \quad \Theta;\emptyset;C_1 \vdash s : P \quad C = C_1 \cup \{OK_n}{C \vdash (D,s)}$$

$$\frac{\Theta;\Gamma;C_1 \vdash x \qquad C = C_1}{\Theta;\Gamma;C \vdash \textbf{endconst}(*x) : \textbf{endco}}$$

$$\frac{\Theta;\Gamma;C_1 \vdash x \qquad \Theta;\Gamma;C_2 \vdash s : P_1 \qquad C =}{\Theta;\Gamma;C \vdash \textbf{const}(*x)s : \textbf{const}}$$

11. Type Inference

$$
\begin{aligned}
&PT_\Theta(f) = \\
&\qquad \textbf{let } \alpha = \Theta(f) \\
&\qquad \textbf{in } (C = \{\alpha \leq \beta\}, \beta) \\
&PT_\Theta(\textbf{skip}) = (\emptyset, 0) \\
&PT_\Theta(s_1; s_2) = \\
&\qquad \textbf{let } (C_1, P_1) = PT_\Theta(s_1) \\
&\qquad\quad (C_2, P_2) = PT_\Theta(s_2) \\
&\qquad \textbf{in } (C_1 \cup C_2 \cup \{P_1; P_2 \leq \beta\}, \beta) \\
&PT_\Theta(*x \leftarrow y) = \\
&\qquad \textbf{let } (C_1, \emptyset) = PT_v(*x) \\
&\qquad\quad (C_2, \emptyset) = PT_v(y) \\
&\qquad \textbf{in } (C_1 \cup C_2, 0) \\
&PT_\Theta(\textbf{free}(x)) = \\
&\qquad \textbf{let } (C_1, \emptyset) = PT_v(x) \\
&\qquad \textbf{in } (C_1, \textbf{free}) \\
&PT_\Theta(\textbf{endconst}(*x)) \qquad\qquad\qquad\qquad\qquad\qquad = \\
&\qquad \textbf{let } (C_1, \emptyset) = PT_v(*x) \\
&\qquad \textbf{in } (C_1, \textbf{endconst}(*x)) \\
&PT_\Theta(\textbf{const}(*x)s) = \\
&\qquad \textbf{let } (C_1, \emptyset) = PT_v(*x) \\
&\qquad \textbf{let } (C_2, P_1) = PT_\Theta(s) \\
&\qquad \textbf{in } (C_1 \cup C_2 \cup P_1 \leq \beta, \textbf{const}(*x)\beta) \\
&PT_\Theta(\textbf{let } x = \textbf{malloc}() \textbf{ in } s) = \\
&\qquad \textbf{let } (C_1, P_1) = PT_v(s) \\
&\qquad \textbf{in } (C_1 \cup \{P_1 \leq \beta\}, \textbf{malloc}; \beta) \\
&PT_\Theta(\textbf{let } x = y \textbf{ in } s) = \\
&\qquad \textbf{let } (C_1, \emptyset) = PT_v(y) \\
&\qquad\quad (C_2, P_1) = PT_\Theta(s) \\
&\qquad \textbf{in } (C_1 \cup C_2 \cup \{P_1 \leq \beta\}, \beta) \\
&PT_\Theta(\textbf{let } x = *y \textbf{ in } s) = \\
&\qquad \textbf{let } (C_1, \emptyset) = PT_v(y) \\
&\qquad\quad (C_2, P_1) = PT_\Theta(s) \\
&\qquad \textbf{in } (C_1 \cup C_2 \cup \{P_1 \leq \beta\}, \beta) \\
&PT_\Theta(\textbf{ifnull } (*x) \textbf{ then } s_1 \textbf{ else } s_2) = \\
&\qquad \textbf{let } (C_1, P_1) = PT_\Theta(s_1) \\
&\qquad\quad (C_2, P_2) = PT_\Theta(s_2) \\
&\qquad\quad (C_3, \emptyset) = PT_v(*x) \\
&\qquad \textbf{in } (C_1 \cup C_2 \cup C_3 \cup \{(*x)(P_1, P_2) \leq \beta\}, \beta)
\end{aligned}
$$

$$
\begin{aligned}
&PT(\langle D, s \rangle) = \\
&\qquad \textbf{let } \Theta = \{f_1 : \alpha_1, \ldots, f_n : \alpha_n\} \\
&\qquad\quad where\ \{f_1, \ldots, f_n\} = dom(D)\ and\ \alpha_1, \ldots, \alpha_n\ are\ fresh \\
&\qquad \textbf{in let } (C_i, P_i) = PT_\Theta(D(f_i))\ for\ each\ i \\
&\qquad \textbf{in let } C_i^{'} = \{\alpha_i \leq P_i\}\ for\ each\ i \\
&\qquad \textbf{in let } (C, P) = PT_\Theta(s) \\
&\qquad \textbf{in } (C_i \cup C_i^{'}) \cup C \cup \{OK(P)\}, P)
\end{aligned}
$$

Figure 4: Type Inference Algorithm