4-2024

Gradual Memory Safety

Jack Phillips
jacphill@nmu.edu

Gradual Memory Safety

By

Jack Phillips

THESIS

Submitted to
Northern Michigan University
In partial fulfillment of the requirements
For the degree of

MASTER OF SCIENCE

Office of Graduate Education and Research

May 2024

SIGNATURE APPROVAL FORM

Gradual Memory Safety

This thesis by Jack Phillips, is recommended for approval by the student's Thesis Committee, the Department Head of the Department of Mathematics and Computer Science, and the Dean of Graduate Education and Research.

_____ Date

Committee Chair: Andy Poe, Professor

_____ Date

First Reader: Randy Appleton, Professor

_____ Date

Second Reader: Dr. Mike Kowalczyk, Professor

_____ Date

Department Head: J.D Phillips, Professor

_____ Date

Dean of Graduate Education and Research: Dr. Lisa Eckert

ABSTRACT

Gradual Memory Safety

By

Jack Phillips

This paper extends the theory of Gradual Types to include memory safe Region-Types and Region-Based Memory Management. It also makes advancements in the capabilities of Region-Based systems. Lastly, it presents the Ŝvejk language and the Haŝek Type System.

DEDICATION

This thesis is dedicated to the memory of Dr. Jeff Horn.

ACKNOWLEDGMENTS

Thank you to my parents for putting up with my shenanigans and encouraging me to pursue my education. Thank you to my professors and teachers who put up with messy assignments. This thesis is the product of a long list of great professors, teachers, and mentors. This thesis uses Chicago Style.

Contents

List of Figures

SYMBOLS AND ABBREVIATIONS

$\mapsto$	Maps To
$\Downarrow$	Evaluates To (Big-Step Semantics)
$\longrightarrow$	Evaluates To (Small-Step Semantics)
$\exists$	Existential Quantifier (There exists)
$\forall$	For All
$\in$	in (contains)
$\notin$	not in (doesn't contain)
$\wedge$	and (operation)
$\vee$	or (operation)
$/$	not (operation)
Γ	Simple Typing Context
F_d, S_d, R_s, E_s	Typing Context 1
$F_d, S_d, H, H_m, E_s, R_s$	Evaluation Context 1
R, E	Typing Context 2
H, V	Evaluation Context 2

1 Introduction

Memory errors are one of the biggest issues concerning security. Microsoft, Google, and Apple report that memory errors cause approximately 60-70% of their security vulnerabilities.[1] Memory-safe languages and approaches have existed for decades, but they all have flaws. In recent years, Memory Safety, without garbage collectors, has become increasingly popular. Region-Based Memory Management is one of the most researched approaches. Traditional Region-Based Memory Systems are verbose and restrictive. They require users to manage regions manually and often require complex annotations. They also impose significant limitations on programs, such as disallowing cycles. This Paper extends Gradual Typing, a recent and popular approach to modern type systems, to Region-Based Memory Management. Specifically, terms typed alpha are dynamic and not stored in Regions, so they have more features and fewer restrictions. Also, I include extensive type inference to massively reduce type annotations that are normally mandatory. Lastly, I formalize it in one language, Ŝvejk, under one cohesive type system, Haŝek. More specifically, my results are: a unified System and Programming Language for Gradual, Safe Memory Management, an approach to references that voids restraints imposed by most region-based memory management type systems, a reduction in the number of required annotations required by Memory Safe Region-Based Type Systems, a rigorous explanation of the static and dynamic Semantics of core Ŝvejk, a Small-step Operational Semantics and a rigorous proof of soundness.

1. "BACK TO THE BUILDING BLOCKS: A Path Toward Secure AND Measurable Software," (accessed: 03.5.2024), https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf.

2 Basic Foundations

Gradually typed systems combine dynamic and static typing under one type system. Also, in Gradually Typed systems, type annotations are optional whenever possible. In essence, a Gradual Type System is a static type system with a special dynamic type. The special dynamic type acts like a supertype of all other typed expressions. In my system, I call this type alpha and type inference works by traversing the type rules.

```
struct str {something:alpha}

let x = 1; # valid, type is infered as an int
x = 1; # valid
x = false; # invalid

let y: alpha = 1;
x = false; # valid
x = [1, 2, 3]; # valid

let z:[alpha] = [1, true, true]; # valid

z = str{something=x}; # not valid, alpha != [alpha]
z[1] = str{something=x}; # valid
```

Figure 1: Gradual Typing

In Figure 1, memory is allocated and deallocated automatically through regions. Region Types obfuscate memory management safely without using a garbage collector. Memory is allocated and freed through Regions, and the Type System guarantees that Region allocation and freeing are safe. Furthermore, Region Type Systems guarantee safe accessing and modifying memory, thus eliminating most memory errors. Simple regions correspond to lexical regions. However, there are more sophisticated dynamic regions where this isn't the case. So, the best way to think about Regions is through their definitions and encoding in the type system and semantics.

2.1 Notation

I describe most notation as it is needed. However, there is some ubiquitous notation that is use throughout, so I describe it here. Let M be some mapping, M: A $\mapsto$ B. If $a_1 \in A$ and $b_1 \in B$, define $M' = M[b_1/a_1]$ such that $M'(a_1) = b_1$, and $M'(a) = M(a)$ where $a \in A$ and $a \neq a_1$. Also, if $a_1 \in A$, $b_1 \in B$ and $M(a_1) = b_1$ define $M' = M/a_1$, such that $a_1 \notin domain(M')$, and $M'(a) = M(a)$ where $a \in A$ and $a \neq a_1$.

Inference Rules are of the form:

$$\textbf{(Inference-Rule)}\ \frac{Premise_1\ Premise_2\ ...\ Premise_n}{Conclusion_1\ Conclusion_2\ ...\ Conclusion_m}$$

Figure 2: Inference Rules

For this paper, an inference rule consists of n premises and m conclusions. You can derive any of the m conclusions if all of the n premises are true. There might be zero premises, in which case the rule is an axiom.

3 Static Semantics

3.1 Foreword

Bertrand Russel devised a Theory of Types to rid formal logic and set theory of paradoxes in the early 20th century.[2] Over time, type theory found its way into programming language design. Type Systems are especially useful for ridding programming languages of errors. Specifically, a type system limits valid programs to programs for which a type can be derived from a set of rules. In other words, a type system only allows programs that type check. Furthermore, the rules ensure that if a program has an error, a type cannot be derived, so the program doesn't type check.

While there are other approaches to handling errors in programming languages, types are simple and precise. They are so simple a compiler can usually incorporate type checking with little overhead, catching errors efficiently before running the program. Traditional type systems are simple, and catch basic errors such as suming a string and an integer. However, type systems are far more powerful and can eliminate more sophisticated and subtle errors like memory errors. With more complex type systems, a formal explanation of the type system is more important to fully describe it and to show that it works. So, I give a formal presentation of the type system and semantics.

Lastly, my idea is original, as is my approach, but I didn't come up with type systems. I also didn't come up with Region-Based Memory Management or Gradual Typing. In short, while I comfortably claim these typing rules are my own, I took inspiration for them from a few places. See references: 2, 3, 4, 5, 8, 9 and 10.

2. Benjamin C. Pierce, "Types and programming languages" (MIT press, 2002).

3.2 Basics

Typing judgements are traditionally of the form $\Gamma \vdash E : t$, meaning in the typing context Γ, E yields type t. However, more complex type systems require more complex typing contexts. For my thesis the typing context is: F_d, S_d, R_s, E_s and typing judgements are of the form $F_d, S_d, R_s, E_s \vdash E : t$, meaning in the typing context of a set of function definitions F_d, a set of structure definitions S_d, a set of regions R_s and a set of environments E_s, E yields type t.

3.3 Types and Typing Context

$F_d : F_{id} \mapsto F_{id}(P_1 : T_2, P_2 : T_2, ...P_n : T_n) \rightarrow T$ (F_{id} is a function name, $P_1, P_2, ...P_n$ are parameters, $T_1, T_2, ...T_n$ are parameter types T is the return type).

$S_d : S_{id} \mapsto S_{id}(A_1 : T_1, A_2 : T_2, ...A_n : T_n)$ (S_{id} is the structure name, $A_1, A_2, ...A_n$ are attribute names, $T_1, T_2, ...T_n$ are attribute types).

$R_s = \{R_1, R_2, ...R_n\}, R_i : id \mapsto t$ (id is an identifier and t is a type)

$E_s = \{E_1, E_2, ...E_m\}, E_i : id \mapsto t$ (id is an identifier and t is a type)

Figure 3: Typing Context

Integers: int

Boolean: bool

Unsafe, Dynamic Type: α

Value Style References: *ref(id, t)

Address Style References: ref(id, t)

List Types: [t] (t is a valid type)

Structure Types: $S_{id}(A_1 : T_1, A_2 : T_2, ...A_n : T_n)$ (S_{id} is the structure name, $A_1, A_2, ...A_n$ are attribute names, $T_1, T_2, ...T_n$ are attribute types).

Function Type: $F_{id}(P_1 : T_2, P_2 : T_2, ...P_n : T_n) \rightarrow T$ (F_{id} is a function name, $P_1, P_2, ...P_n$ are parameters, $T_1, T_2, ...T_n$ are parameter types T is the return type).

Safe Statement: $void \times t$ (t is a valid type capturing the return)

Unsafe Statement: $\alpha \times t$ (t is a valid type capturing the return)

Figure 4: Types of Types One

Definition A type t is safe if:

t = int or bool

t = $*ref(id, t')$ or t = $ref(id, t')$ and t' is safe

t = $[t']$ and t' is safe

$S_d(t) = S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$ and t_i is safe $\forall t \in t_1, t_2, ...t_n$

t = $void \times T$ and T is safe

Definition A type t is unsafe if:

t = α

t = $*ref(id, t')$ or t = $ref(id, t')$ and t' is unsafe

t = $[t']$ and t' is unsafe

$S_d(t) = S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$ and $\exists t_i \in t_1, t_2, ...t_n$ such that t_i is unsafe

t = $\alpha \times T$

Definition: Subtyping: t1 <: t2, t1 is a subtype of t2

Figure 5: Safe, Unsafe and Subtyping Definition

3.4 Definitions and Functions

Definition: closeScope $closeScope(R_s, E_s) = R'_s, E'_s$

$R'_s = R_s$ without $R \in R_s$, where $R' \succeq R,\ \forall R' \in R_s$

$E'_s = E_s$ without $E \in E_s$, where $E' \succeq E,\ \forall E' \in E_s$

Definition: newScope

$newScope(Rs) = R'_s$ where $Rs' = R_s$ with some new Region $R = \emptyset$ such that $R' \succeq R, \forall R' \in R_s$

$newScope(Es) = E'_s$ where $Es' = E_s$ with some new Environment $E = \emptyset$ such that $E' \succeq E, \forall E' \in E_s$

Definition: declare

$declare(R_s, id, t) = R'_s,\ Rs' = R_s$ where $R[t/id]$ for $R = find(R_s, id)$

$declare(E_s, id, t) = E'_s,\ Es' = E_s$ where $E[t/id]$ for $E = find(E_s, id)$

Definition: find

$find(R_s, id) = R$, where $R' \succeq R,\ \forall R' \in R_s$ such that $R'(id) = t$ and $R(id) = t$

$find(E_s, id) = E$, where $E' \succeq E,\ \forall E' \in E_s$ such that $E'(id) = t$ and $E(id) = t$

Definition: currentScope

$currentScope(R_s, id) = R$, where $R' \succeq R,\ \forall R' \in R_s$

$currentScope(E_s, id) = E$, where $E' \succeq E,\ \forall E' \in E_s$

Figure 6: Type System Function Definitions

3.5 Type System

Simple Expressions

$$\textbf{(TS-E-1)}\ \frac{i \text{ is a valid integer}}{F_d, S_d, R_s, E_s \vdash i : int}$$

$$\frac{b \text{ is a valid boolean}}{F_d, S_d, R_s, E_s \vdash b : bool}\ \textbf{(TS-E-2)}$$

$$\textbf{(TS-E-3)}\ \frac{F_d, S_d, R_s, E_s \vdash b_1 : bool \qquad F_d, S_d, R_s, E_s \vdash b_2 : bool \quad \text{op} \in \{\text{and}, \text{or}\}}{F_d, S_d, R_s, E_s \vdash b_1\ op\ b_2 : bool}$$

$$\textbf{(TS-E-3a)}\ \frac{F_d, S_d, R_s, E_s \vdash b_1 : t_1 \qquad F_d, S_d, R_s, E_s \vdash b_2 : t_2 \quad (t_1 = \alpha \wedge t_2 = bool) \vee (t_2 = \alpha \wedge t_1 = bool) \vee t_1, t_2 = \alpha \quad \text{op} \in \{\text{and}, \text{or}\}}{F_d, S_d, R_s, E_s \vdash b_1\ op\ b_2 : \alpha}$$

$$\frac{F_d, S_d, R_s, E_s \vdash b : bool}{F_d, S_d, R_s, E_s \vdash \text{!b : bool}}\ \textbf{(TS-E-4)}$$

$$\textbf{(TS-E-4a)}\ \frac{F_d, S_d, R_s, E_s \vdash b : \alpha}{F_d, S_d, R_s, E_s \vdash !b : \alpha}$$

Figure 7: Simple Expression Typing Rules A

$$\frac{F_d, S_d, R_s, E_s \vdash i_1 : int \qquad F_d, S_d, R_s, E_s \vdash i_2 : int \qquad \text{op} \in \{+,-,^*\}}{F_d, S_d, R_s, E_s \vdash i_1 \ op \ i_2 : int} \textbf{(TS-E-5)}$$

$$\textbf{(TS-E-5a)} \ \frac{F_d, S_d, R_s, E_s \vdash i_1 : t_1 \qquad F_d, S_d, R_s, E_s \vdash i_2 : t_2 \qquad \text{op} \in \{+,-,^*\} \qquad (t_1 = \alpha \wedge t_2 = int) \vee (t_2 = \alpha \wedge t_1 = int) \vee t_1, t_2 = \alpha}{F_d, S_d, R_s, E_s \vdash i_1 \ op \ i_2 : \alpha}$$

$$\frac{F_d, S_d, R_s, E_s \vdash i_1 : int \qquad F_d, S_d, R_s, E_s \vdash i_2 : int \qquad op \in \{<,>,==\}}{F_d, S_d, R_s, E_s \vdash i_1 \ op \ i_2 : bool} \textbf{(TS-E-6)}$$

$$\frac{F_d, S_d, R_s, E_s \vdash i_1 : t_1 \qquad F_d, S_d, R_s, E_s \vdash i_2 : t_2 \qquad op \in \{<,>,==\} \qquad (t_1 = \alpha \wedge t_2 = int) \vee (t_2 = \alpha \wedge t_1 = int) \vee t_1, t_2 = \alpha}{F_d, S_d, R_s, E_s \vdash i_1 \ op \ i_2 : \alpha} \textbf{(TS-E-6a)}$$

Figure 8: More Simple Expression Typing Rules A

Variable Declaration

$$\textbf{(TS-D-1)}\ \frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : \ t \vee e = nil \\ t \text{ is safe} \qquad t \neq ref(id', t')\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash let\ id : t\ =\ e;\ : void \times void \\ F_d, S_d, R_s, E_s \vdash let\ id\ =\ e;\ : void \times void\end{array}}$$

$$\frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : \ ref(id', t) \\ id \neq id' \qquad t \text{ is safe}\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash let\ id : t\ =\ e;\ : void \times void \\ F_d, S_d, R_s, E_s \vdash let\ id\ =\ e;\ : void \times void\end{array}}\ \textbf{(TS-D-2)}$$

$$\textbf{(TS-D-1a)}\ \frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : t \\ t \text{ is unsafe} \qquad t \neq ref(id', t')\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash let\ id : t\ =\ e;\ : \alpha \times void \\ F_d, S_d, R_s, E_s \vdash let\ id\ =\ e;\ : \alpha \times void\end{array}}$$

$$\frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : ref(id', t) \\ t \text{ is unsafe} \qquad id \neq id'\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash let\ id : t\ =\ e;\ : \alpha \times void \\ F_d, S_d, R_s, E_s \vdash let\ id\ =\ e;\ : \alpha \times void \\ F_d, S_d, R_s, E_s \vdash let\ id : \alpha\ =\ e;\ : \alpha \times void\end{array}}\ \textbf{(TS-D-2a)}$$

Figure 9: Variable Declaration Typing Rules

$$\textbf{(TS-D-1b)}\ \frac{\begin{array}{c} id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : t \vee e = nil \\ t \neq ref(id', t') \end{array}}{F_d, S_d, R_s, E_s \vdash let\ id : \alpha\ =\ e;\ : \alpha \times void}$$

Figure 10: More Variable Declaration Typing Rules

Assignment

$$\textbf{(TS-A-1)}\ \frac{\begin{array}{c} find(R_s, id) = R \qquad R(id) \ = \ t \\ F_d, S_d, R_s, E_s \vdash e : \ t \vee e = nil \end{array}}{F_d, S_d, R_s, E_s \vdash id \ = \ e \ ; : void \times void}$$

$$\frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash e : ref(id', t) \qquad id \neq id' \\ find(R_s, id) = R \qquad find(R_s, id') = R' \\ R(id) \ = R'(id') \ = \ t \qquad R' \succeq R \end{array}}{F_d, S_d, R_s, E_s \vdash id \ = \ e \ ; : void \times void}\ \textbf{(TS-A-2)}$$

$$\textbf{(TS-A-1a)}\ \frac{\begin{array}{c} find(E_s, id) = E \\ F_d, S_d, R_s, E_s \vdash e : t \vee e = nil \end{array}}{F_d, S_d, R_s, E_s \vdash id \ = \ e \ ; : \alpha \times void}$$

$$\frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash ref(id', t) \qquad id \neq id' \\ find(E_s, id) = E \qquad find(E_s, id') = E' \end{array}}{F_d, S_d, R_s, E_s \vdash id \ = \ e \ ; : \alpha \times void}\ \textbf{(TS-V-A-2a)}$$

Reference

$$\textbf{(TS-R-1)}\ \frac{\begin{array}{c} find(R_s, id) = R \wedge R(id) \ = \ t \ \vee \\ find(E_s, id) = E \wedge E(id) = t \end{array}}{\begin{array}{c} F_d, S_d, R_s, E_s \vdash id \ : *ref(id, t) \\ F_d, S_d, R_s, E_s \vdash \&id \ : ref(id, t) \\ F_d, S_d, R_s, E_s \vdash ref\{id, a\} \ : ref(id, t) \end{array}}\ \frac{}{*ref(id, t) <: t}\ \textbf{(TS-R-2)}$$

Figure 11: Other Variable Typing Rules A

Scope

$$\textbf{(TS-S-1)}\ \frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : \ t \\ t \text{ is safe} \qquad t \neq ref(id', t') \\ R'_s = declare(R_s, id, t) \\ F_d, S_d, R'_s, E \vdash S : t'' \times t'''\end{array}}{\begin{array}{c}F_d, S_d, R_s, E \vdash let\ id : t\ =\ e; S : t'' \times t''' \\ F_d, S_d, R_s, E \vdash let\ id\ =\ e; S : t'' \times t'''\end{array}}$$

$$\frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : \ ref(id', t) \\ id \neq id' \qquad t \text{ is safe} \\ R'_s = declare(R_s, id, t) \\ F_d, S_d, R'_s, E \vdash S : t' \times t''\end{array}}{\begin{array}{c}F_d, S_d, R_s, E \vdash let\ id : t\ =\ e; S : t' \times t'' \\ F_d, S_d, R_s, E \vdash let\ id\ =\ e; S : t' \times t''\end{array}}\ \textbf{(TS-S-2)}$$

$$\textbf{(TS-S-1a)}\ \frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : t \\ t \text{ is unsafe} \qquad t \neq ref(id', t') \\ E'_s = declare(E_s, id, t) \\ F_d, S_d, R_s, E'_s \vdash S : t'' \times t'''\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash let\ id : t\ =\ e; S\ : \alpha \times t''' \\ F_d, S_d, R_s, E_s \vdash let\ id\ =\ e; S\ : \alpha \times t''' \\ F_d, S_d, R_s, E \vdash let\ id\ =\ nil; S : \alpha \times t'''\end{array}}$$

$$\frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : ref(id', t) \\ t \text{ is unsafe} \qquad id \neq id' \\ E'_s = declare(E_s, id, t) \\ F_d, S_d, R_s, E'_s \vdash S : t' \times t''\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash let\ id : t\ =\ e; S\ : \alpha \times t'' \\ F_d, S_d, R_s, E_s \vdash let\ id\ =\ e; S\ : \alpha \times t''\end{array}}\ \textbf{(TS-S-2a)}$$

Figure 12: Scope Typing Rules

$$\textbf{(TS-S-2b)}\ \frac{\begin{array}{c} id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : ref(id', t) \\ t \text{ is unsafe} \qquad id \neq id' \\ E'_s = declare(E_s, id, \alpha) \\ F_d, S_d, R_s, E'_s \vdash S : t' \times t'' \end{array}}{F_d, S_d, R_s, E_s \vdash let\ id : \alpha\ =\ e; S\ : \alpha \times t''}$$

$$\frac{\begin{array}{c} id \text{ is a valid identifier} \\ id \notin currentScope(R_s) \cup currentScope(E_s) \\ F_d, S_d, R_s, E_s \vdash e : t \vee e = nil \\ t \neq ref(id', t') \\ E'_s = declare(E_s, id, \alpha) \\ F_d, S_d, R_s, E_s \vdash S : t'' \times t''' \end{array}}{F_d, S_d, R_s, E_s \vdash let\ id : \alpha\ =\ e; S\ : \alpha \times t'''}\ \textbf{(TS-S-1b)}$$

Figure 13: More Scope Typing Rules

Control Flow

$$\textbf{(TS-C-1)}\ \frac{\begin{array}{c}E'_s = newScope(E_s)\\ R'_s = newScope(R_s)\\ F_d, S_d, R_s, E_s \vdash b\ : bool\\ F_d, S_d, R'_s, E'_s \vdash S_1\ : void \times t\\ F_d, S_d, R'_s, E'_s \vdash S_2\ : void \times t\end{array}}{F_d, S_d, R_s, E_s \vdash if\ b\ \{S_1\}\ else\ \{S_2\} : void \times t}$$

$$\frac{\begin{array}{c}E'_s = newScope(E_s)\\ R'_s = newScope(R_s)\\ F_d, S_d, R_s, E_s \vdash b\ : t_1\\ F_d, S_d, R'_s, E'_s \vdash S_1\ : t_2 \times t'\\ F_d, S_d, R'_s, E'_s \vdash S_2\ : t_3 \times t''\\ (t_1 = bool \vee t_1 = \alpha)\ \wedge t' \neq t''\end{array}}{F_d, S_d, R_s, E_s \vdash if\ b\ \{S_1\}\ else\ \{S_2\} : \alpha \times \alpha}\ \textbf{(TS-C-1a)}$$

$$\textbf{(TS-C-1b)}\ \frac{\begin{array}{c}E'_s = newScope(E_s)\\ R'_s = newScope(R_s)\\ F_d, S_d, R_s, E_s \vdash b\ : t_1\\ F_d, S_d, R'_s, E'_s \vdash S_1\ : t_2 \times t'\\ F_d, S_d, R'_s, E'_s \vdash S_2\ : t_2 \times t'\\ t_1 = \alpha \vee (t_1 = bool \wedge t_2 = \alpha)\end{array}}{F_d, S_d, R_s, E_s \vdash if\ b\ \{S_1\}\ else\ \{S_2\} : \alpha \times t'}$$

$$\frac{\begin{array}{c}E'_s = newScope(E_s)\\ R'_s = newScope(R_s)\\ F_d, S_d, R_s, E_s \vdash b\ : bool\\ F_d, S_d, R'_s, E'_s \vdash S\ : void \times t\end{array}}{F_d, S_d, R_s, E_s \vdash while\ b\ \{S\} : void \times t}\ \textbf{(TS-C-2)}$$

$$\frac{\begin{array}{c}E'_s = newScope(E_s)\\ R'_s = newScope(R_s)\\ F_d, S_d, R_s, E_s \vdash b\ : t\\ F_d, S_d, R'_s, E'_s \vdash S\ : t' \times t''\\ t = \alpha \vee (t = bool \wedge t' = \alpha)\end{array}}{F_d, S_d, R_s, E_s \vdash while\ b\ \{S\} : \alpha \times t''}\ \textbf{(TS-C-2a)}$$

Figure 14: Control Flow Statement Typing Rules

$$\textbf{(TS-C-3)}\ \frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash S_1\ : t \times t' \\ F_d, S_d, R_s, E_s \vdash S_2\ : t \times t' \\ S_1 \text{ isn't a let statement} \end{array}}{F_d, S_d, R_s, E_s \vdash S_1\ S_2 : t \times t'}$$

$$\frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash S_1\ : t_1 \times t' \\ F_d, S_d, R_s, E_s \vdash S_2\ : t_2 \times t'' \\ t_1 = \alpha \vee t_2 = \alpha \vee t' \neq t'' \\ S_1 \text{ isn't a let statement} \end{array}}{F_d, S_d, R_s, E_s \vdash S_1\ S_2 : \alpha \times \alpha}\ \textbf{(TS-C-3a)}$$

Figure 15: More Control Flow Statement Typing Rules

Other Statements

$$\textbf{(TS-R-1)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e : t \\ t \text{ is safe}\end{array}}{F_d, S_d, R_s, E_s \vdash return\ e;\ : void \times t}$$

$$\frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e : t \\ t \text{ is unsafe}\end{array}}{F_d, S_d, R_s, E_s \vdash return\ e;\ : \alpha \times t}\ \textbf{(TS-R-1a)}$$

$$\textbf{(TS-O-1)}\ \frac{F_d, S_d, R_s, E_s \vdash e : t \qquad t \text{ is safe}}{F_d, S_d, R_s, E_s \vdash e; : void \times void}$$

$$\frac{F_d, S_d, R_s, E_s \vdash e : t \qquad t \text{ is unsafe}}{F_d, S_d, R_s, E_s \vdash e; : \alpha \times void}\ \textbf{(TS-O-1a)}$$

$$\frac{F_d, S_d, R_s, E_s \vdash e : ref(id', t) \qquad t \text{ is unsafe}}{F_d, S_d, R_s, E_s \vdash free\ (e); : \alpha \times void}\ \textbf{(TS-O-2)}$$

Figure 16: Other Statement Typing Rules

Function Definitions

$$\textbf{(TS-F-1)}\ \frac{\begin{array}{c} F_d(F_{id}) = F_{id}(P_1 : T_1, P_2 : T_2, ...P_n : T_n) \to T_r \\ P_1,\ P_2,\ ...,\ P_n \text{ are unique} \\ \forall t \in T_1,\ T_2,\ ...,\ T_n,\ T_r : t \text{ is safe} \\ R' = \{P_1 : T_1, P_2 : T_2, ..., P_n : T_n\} \\ R'_s = \{R'\} \\ F_d, S_d, R'_s, E_s \vdash S_1\ S_2\ ...\ S_j\ return\ e; : void \times T' \\ T' = T_r \wedge T' \neq ref(id, T'') \\ j, n \geq 0 \end{array}}{\begin{array}{c} F_d, S_d, R_s, E_s \vdash def\ F_{id}(P_1 : T_1,\ P_2 : T_2, ...\ P_n : T_n) \to \\ T_r\ \{S_1\ S_2\ ...\ S_j\ return\ e;\} : F_{id}(P_1 : T_1,\ P_2 : T_2, ...\ P_n : T_n) \to T_r \end{array}}$$

$$\textbf{(TS-F-1a)}\ \frac{\begin{array}{c} F_d(F_{id}) = F_{id}(P_1 : T_1, P_2 : T_2, ...P_n : T_n) \to T_r \\ P_1,\ P_2,\ ...,\ P_n \text{ are unique} \\ \forall t \in T_{a1},\ T_{a2},\ ...,\ T_{ab} : t \text{ is safe} \\ \forall t \in T_{c1},\ T_{c2},\ ...,\ T_{cd}, T_r : t \text{ is unsafe} \\ b, d, n \geq 0 \qquad b + d = n \\ T_{a1},\ T_{a2},\ ...,\ T_{ab} \cup T_{c1},\ T_{c2},\ ...,\ T_{cd} = T_1,\ T_2,\ ...,\ T_n \\ R' = \{P_{a1} : T_{a1}, P_{a2} : T_{a2}, ..., P_{ab} : T_{ab}\} \\ E' = \{P_{c1} : T_{c1}, P_{c2} : T_{c2}, ..., P_{cd} : T_{cd}\} \\ R'_s = \{R'\} \qquad E'_s = \{E'\} \\ F_d, S_d, R'_s, E'_s \vdash S_1\ S_2\ ...\ S_j\ return\ e; : \alpha \times T'_r \\ T_r = \alpha \vee T'_r = T_r \end{array}}{\begin{array}{c} F_d, S_d, R_s, E \vdash def\ F_{id}(P_1 : T_1,\ P_2 : T_2, ...\ P_n : T_n) \to \\ T_r\ \{S_1\ S_2\ ...\ S_j\ return\ e;\} : F_{id}(P_1 : T_1,\ P_2 : T_2, ...\ P_n : T_n) \to T_r \end{array}}$$

Figure 17: Function Definition Typing Rules

Function Call

$$\textbf{(TS-F-2)}\ \frac{\begin{array}{c} F_d(F_{id}) = F_{id}(P_1 : T_1, P_2 : T_2, ...P_n : T_n) \to T \\ (F_d, S_d, R_s, E_s \vdash arg_1 : T_1) \vee (F_d, S_d, R_s, E_s \vdash arg_1 : T_1' \wedge T_1 = \alpha) \vee \\ (F_d, S_d, R_s, E_s \vdash arg_1 : \mathit{ref}(id, T_1)) \vee arg_1 = nil \\ (F_d, S_d, R_s, E_s \vdash arg_2 : T_2) \vee (F_d, S_d, R_s, E_s \vdash arg_2 : T_2' \wedge T_2 = \alpha) \vee \\ (F_d, S_d, R_s, E_s \vdash arg_2 : \mathit{ref}(id, T_2)) \vee arg_2 = nil \\ \vdots \\ (F_d, S_d, R_s, E_s \vdash arg_n : T_n) \vee (F_d, S_d, R_s, E_s \vdash arg_n : T_n' \wedge T_n = \alpha) \vee \\ (F_d, S_d, R_s, E_s \vdash arg_n : \mathit{ref}(id, T_n)) \vee arg_n = nil \end{array}}{F_d, S_d, R_s, E_s \vdash F_{id}(arg_1,\ arg_2, ...\ arg_n) : F_{id}(P_1 : T_1, P_2 : T_2, ...P_n : T_n) \to T}$$

Figure 18: Function Call Typing Rules

Structure Definition

$$\textbf{(TS-St-1)}\ \frac{\begin{array}{c} S_d(S_{id}) = S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \\ A_1,\ A_2,\ ...,\ A_n \textit{ are unique} \\ \forall t \in T_1,\ T_2,\ ...,\ T_n : t \text{ is safe} \vee t \text{ is unsafe} \\ n \geq 1 \end{array}}{\begin{array}{c} F_d, S_d, R_s, E_s \vdash \mathit{struct}\ S_{id}\{A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n\} : \\ S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \end{array}}$$

Figure 19: Structure Definition Typing Rules

Structure Literal

$$\textbf{(TS-St-2)}\ \frac{\begin{array}{c} S_d(S_{id}) = S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \\ F_d, S_d, R_s, E_s \vdash e_1 : T_1' \wedge T_1' = T_1 \vee T_1 = \alpha \\ F_d, S_d, R_s, E_s \vdash e_2 : T_2' \wedge T_2' = T_2 \vee T_2 = \alpha \\ \vdots \\ F_d, S_d, R_s, E_s \vdash e_n : T_n' \wedge T_n' = T_n \vee T_n = \alpha \end{array}}{F_d, S_d, R_s, E_s \vdash S_{id}\{A_1 = e_1,\ A_2 = e_2, ...,\ A_n = e_n\} : S_{id}}$$

Figure 20: Structure Literal Typing Rules

Structure Assignment

$$\textbf{(TS-St-3)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id', S_{id}) \\ S_d(S_{id}) = S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \\ \forall t \in T_1, T_2, ...\ T_n : t \text{ is safe} \qquad id = A_i \in A_1, A_2, ...A_n \\ F_d, S_d, R_s, E_s \vdash e_2 : T_i \qquad T_i \neq ref(id'', t)\end{array}}{F_d, S_d, R_s, E_s \vdash e_1.id = e_2 : void \times void}$$

$$\textbf{(TS-St-3a)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id', S_{id}) \\ S_d(S_{id}) = S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \\ \forall t \in T_1, T_2, ...\ T_n : t \text{ is safe} \qquad id = A_i \in A_1, A_2, ...A_n \\ F_d, S_d, R_s, E_s \vdash e_2 : ref(id'', T_i) \\ find(R_s, id') = R \qquad find(R_s, id'') = R' \qquad R' \succeq R\end{array}}{F_d, S_d, R_s, E_s \vdash e_1.id = e_2 : void \times void}$$

$$\textbf{(TS-St-3b)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id', S_{id}) \\ S_d(S_{id}) = S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \\ id = A_i \in A_1, A_2, ...A_n \qquad T_i \text{ is unsafe} \\ F_d, S_d, R_s, E_s \vdash e_2 : T_i' \qquad T_i' = T_i \vee T_i = \alpha \qquad T_i' \neq ref(id'', t)\end{array}}{F_d, S_d, R_s, E_s \vdash e_1.id = e_2 : \alpha \times void}$$

$$\textbf{(TS-St-3c)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id', S_{id}) \\ S_d(S_{id}) = S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \\ id = A_i \in A_1, A_2, ...A_n \qquad T_i \text{ is unsafe} \\ F_d, S_d, R_s, E_s \vdash e_2 : ref(id', T_i') \qquad T_i = \alpha \vee T_i' = T_i\end{array}}{F_d, S_d, R_s, E_s \vdash e_1.id = e_2 : \alpha \times void}$$

$$\textbf{(TS-St-3d)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id', \alpha) \\ F_d, S_d, R_s, E_s \vdash e_2 : t \\ t \neq ref(id'', t')\end{array}}{F_d, S_d, R_s, E_s \vdash e_1.id = e_2 : \alpha \times void}$$

Figure 21: Structure Assignment Typing Rules

Structure Assignment

$$\frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id', \alpha) \\ F_d, S_d, R_s, E_s \vdash e_2 : ref(id'', t) \\ t \text{ is unsafe}\end{array}}{F_d, S_d, R_s, E_s \vdash e_1.id = e_2 : \alpha \times void} \textbf{(TS-St-3e)}$$

Figure 22: More Structure Assignment Typing Rules

Structure Attribute References

$$\textbf{(TS-St-4)} \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id', S_{id}) \\ S_d(S_{id}) = S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \\ id = A_i \in A_1, A_2, ...A_n\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1.id : *ref(id', T_i) \\ F_d, S_d, R_s, E_s \vdash \&e_1.id : ref(id', T_i)\end{array}}$$

$$\textbf{(TS-St-5)} \frac{F_d, S_d, R_s, E_s \vdash e_1 : *ref(id', \alpha)}{F_d, S_d, R_s, E_s \vdash e_1.id : \alpha}$$

Figure 23: Structure Reference Typing Rules

Global Definitions

$$
\begin{array}{c}
T_1' = F_{id1}(P_{11} : T_{11},\ P_{12} : T_{12}, ...,\ P_{1n_1} : T_{1n_1}) \to T_{r1} \\
T_2' = F_{id2}(P_{21} : T_{21},\ P_{22} : T_{22}, ...,\ P_{2n_2} : T_{2n_2}) \to T_{r2} \\
\vdots \\
T_j' = F_{idj}(P_{j1} : T_{j1},\ P_{j2} : T_{j2}, ...,\ P_{jn_j} : T_{jn_j}) \to T_{rj} \\
T_{j+1}' = S_{id1}(A_{11} : T_{11}'',\ A_{12} : T_{12}'', ...,\ A_{1m_1} : T_{1m_1}'') \\
T_{j+2}' = S_{id2}(A_{21} : T_{21}'',\ A_{22} : T_{22}'', ...,\ A_{2m_2} : T_{2m_2}'') \\
\vdots \\
T_{j+k}' = S_{idk}(A_{k1} : T_{k1}'',\ A_{k2} : T_{k2}'', ...,\ A_{km_k} : T_{km_k}'') \\
F_{d1} = F_d[T_1' \ / F_{id1}][T_2' \ / F_{id2}] \ ...,\ [T_j' \ / F_{idj}] \\
S_{d1} = S_d[T_{j+1}' \ / S_{id1}][T_{j+2}' \ / S_{id2}] \ ...,\ [T_{j+k}' \ / S_{idk}] \\
F_{d1}, S_{d1}, R_s, E_s \vdash def\ F_{id1}(P_{11} : T_{11},\ P_{12} : T_{12}, ...,\ P_{1n_1} : T_{1n_1}) \to T_{r1}\ \{S_{11}S_{12}...S_{1x_1}\ return\ e_1;\} : T_1' \\
F_{d1}, S_{d1}, R_s, E_s \vdash def\ F_{id2}(P_{21} : T_{21},\ P_{22} : T_{22}, ...,\ P_{2n_2} : T_{2n_2}) \to T_{r2}\ \{S_{21}S_{22}...S_{2x_2}\ \ return\ e_2;\} : T_2' \\
\vdots \\
F_{d1}, S_{d1}, R_s, E_s \vdash def\ F_{idj}(P_{j1} : T_{j1},\ P_{j2} : T_{j2}, ...,\ P_{jn_j} : T_{jn_j}) \to T_{rj}\ \{S_{j1}S_{j2}...S_{jx_j}\ \ return\ e_j;\} : T_j' \\
F_{d1}, S_{d1}, R_s, E_s \vdash struct\ S_{id1}\{A_{11} : T_{11}'',\ A_{12} : T_{12}'', ...,\ A_{1m_1} : T_{1m_1}''\} : T_{j+1}' \\
F_{d1}, S_{d1}, R_s, E_s \vdash struct\ S_{id2}\{A_{21} : T_{21}'',\ A_{22} : T_{22}'', ...,\ A_{2m_2} : T_{2m_2}''\} : T_{j+2}' \\
\vdots \\
F_{d1}, S_{d1}, R_s, E_s \vdash struct\ S_{idk}\{A_{k1} : T_{k1}'',\ A_{k2} : T_{k2}'', ...,\ A_{km_k} : T_{km_k}''\}\ S : T_{j+k}'
\end{array}
$$

(TS-G-1)

$$
\frac{\begin{array}{c} F_{d1}, S_{d1}, R_s, E_s \vdash\ S : t \\ 0 \leq j, k \\ 0 \leq n_1, n_2, ...n_j \\ 0 \leq x_1, x_2, x_3, ..., x_j \\ 0 \leq m_1, m_2, ...m_k \\ S_{id} \text{ are unique} \qquad F_{id} \text{ are unique} \\ A_i,\ P_i \text{ are unique in the context of their definitions} \end{array}}{\begin{array}{c} F_d, S_d, R_s, E_s \vdash def\ F_{id1}(P_{11} : T_{11},\ P_{12} : T_{12}, ...,\ P_{1n_1} : T_{1n_1}) \to T_{r1}\ \{S_{11}S_{12}...S_{1x_1}\ return\ e_1;\} \\ def\ F_{id2}(P_{21} : T_{21},\ P_{22} : T_{22}, ...,\ P_{2n_2} : T_{2n_2}) \to T_{r2}\ \{S_{21}S_{22}...S_{2x_1}\ return\ e_2;\} \\ \vdots \\ def\ F_{idj}(P_{j1} : T_{j1},\ P_{j2} : T_{j2}, ...,\ P_{jn_j}\ T_{jn_j}) \to T_{rj}\ \{S_{j1}S_{j2}...S_{jx_j}\ return\ e_j;\} \\ struct\ S_{id1}\{A_{11} : T_{11}'',\ A_{12} : T_{12}'', ...,\ A_{1m_1} : T_{1m_1}''\} \\ struct\ S_{id2}\{A_{21} : T_{21}'',\ A_{22} : T_{22}'', ...,\ A_{2m_2} : T_{2m_2}''\} \\ \vdots \\ struct\ S_{idk}\{A_{k1} : T_{k1}'',\ A_{k2} : T_{k2}'', ...,\ A_{km_k} : T_{km_k}''\}\ S : t \end{array}}
$$

Figure 24: Global Definitions Typing Rules A

List Literals

$$\textbf{(TS-L-1)}\ \frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash e_1 : t \\ F_d, S_d, R_s, E_s \vdash e_1 : t \\ \vdots \\ F_d, S_d, R_s, E_s \vdash e_n : t \\ t \neq ref(id, t') \end{array}}{F_d, S_d, R_s, E_s \vdash [e_1, e_2, ...e_n] : [t]}$$

$$\frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash e_1 : t_1 \\ F_d, S_d, R_s, E_s \vdash e_1 : t_2 \\ \vdots \\ F_d, S_d, R_s, E_s \vdash e_n : t_n \\ \forall t \in t_1, t_2, ...t_n : t \neq ref(id, t') \\ \exists t, t' \in t_1, t_2, ...t_n : t \neq t' \vee t_i \text{ is unsafe} \end{array}}{F_d, S_d, R_s, E_s \vdash [e_1, e_2, ...e_n] : [\alpha]}\ \textbf{(TS-L-1a)}$$

Figure 25: List Literal Typing Rules

List Accessing and Modifying

$$\textbf{(TS-L-O-1)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, t) \\ F_d, S_d, R_s, E_s \vdash e_2 : t' \\ t' \neq ref(id', t'') \\ t = \alpha \vee (t \text{ is unsafe} \wedge t = [t'])\end{array}}{F_d, S_d, R_s, E_s \vdash e_1 << e_2; : \alpha \times void}$$

$$\frac{F_d, S_d, R_s, E_s \vdash e : *ref(id, \alpha)}{F_d, S_d, R_s, E_s \vdash e! : \alpha}\ \textbf{(TS-L-2)}$$

$$\frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e : *ref(id, [t]) \\ t \neq ref(id', t') \qquad t \text{ is unsafe}\end{array}}{F_d, S_d, R_s, E_s \vdash e! : t}\ \textbf{(TS-L-2a)}$$

$$\textbf{(TS-L-3)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, [t]) \\ F_d, S_d, R_s, E_s \vdash e_2 : t' \\ t \neq ref(id', t'') \qquad (t' = int \wedge t \text{ is safe}) \vee \\ ((t' = int \vee t' = \alpha) \wedge t \text{ is unsafe})\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1[e_2] : *ref(id, t) \\ F_d, S_d, R_s, E_s \vdash \&e_1[e_2] : ref(id, t)\end{array}}$$

$$\frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, t) \\ F_d, S_d, R_s, E_s \vdash e_2 : t' \\ t' = int \vee t' = \alpha \\ t = \alpha \vee (t' = \alpha \wedge t \text{ is safe} \wedge t = [t''])\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1[e_2] : *ref(id, \alpha) \\ F_d, S_d, R_s, E_s \vdash \&e_1[e_2] : ref(id, \alpha)\end{array}}\ \textbf{(TS-L-3a)}$$

$$\textbf{(TS-L-4)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, [t]) \\ F_d, S_d, R_s, E_s \vdash e_2 : int \\ F_d, S_d, R_s, E_s \vdash e_3 : t \\ t \text{ is safe} \qquad t \neq ref(id', t')\end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2] = e_3 : void \times void}$$

Figure 26: List Accessing and Modifying Typing Rules

$$\frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, t) \\ F_d, S_d, R_s, E_s \vdash e_2 : t' \\ F_d, S_d, R_s, E_s \vdash e_3 : t'' \\ t' = \alpha \vee t' = int \\ t = \alpha \vee t = [t'''] \wedge t'' = t''' \wedge \\ t \text{ is safe} \wedge t' = \alpha \vee t \text{ is unsafe} \end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2] = e_3 : \alpha \times void} \textbf{ (TS-L-4a)}$$

$$\textbf{(TS-L-5)} \frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, [t]) \\ F_d, S_d, R_s, E_s \vdash e_2 : int \\ F_d, S_d, R_s, E_s \vdash e_3 : ref(id', t) \\ t \text{ is safe} \\ find(R_s, id) = R \qquad find(R_s, id') = R' \\ R' \succeq R \end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2] = e_3 : void \times void}$$

$$\frac{\begin{array}{c} F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, t) \\ F_d, S_d, R_s, E_s \vdash e_2 : t' \\ F_d, S_d, R_s, E_s \vdash e_3 : ref(id', t'') \\ find(E_s, id) = E \qquad find(E_s, id') = E' \\ t' = \alpha \vee int \\ t = \alpha \vee (t \text{ is unsafe} \wedge t = [t'']) \end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2] = e_3 : \alpha \times void} \textbf{ (TS-L-5a)}$$

Figure 27: More List Accessing and Modifying Typing Rules

Other List Operations

$$\textbf{(TS-L-6)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e : *ref(id, [t]) \\ t \text{ is safe}\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash \#e : int \\ F_d, S_d, R_s, E_s \vdash e\# : int\end{array}}$$

$$\frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e : *ref(id, t) \\ t = \alpha \vee (t \text{ is unsafe} \wedge t = [t'])\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash \#e : \alpha \\ F_d, S_d, R_s, E_s \vdash e\# : \alpha\end{array}}\ \textbf{(TS-L-6a)}$$

$$\textbf{(TS-L-7)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, [t]) \\ F_d, S_d, R_s, E_s \vdash e_2 : ref(id', [t])\end{array}}{F_d, S_d, R_s, E_s \vdash e_1 + e_2 : [t]}$$

$$\frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, t) \\ F_d, S_d, R_s, E_s \vdash e_2 : ref(id', t') \\ (t = \alpha \wedge t' = [t'']) \vee (t = [t''] \wedge t' = \alpha) \vee t' = t'' = \alpha\end{array}}{F_d, S_d, R_s, E_s \vdash e_1 + e_2 : \alpha}\ \textbf{(TS-L-7a)}$$

$$\textbf{(TS-L-8)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, [t]) \\ F_d, S_d, R_s, E_s \vdash e_2 : int \\ F_d, S_d, R_s, E_s \vdash e_3 : int \\ t \text{ is safe}\end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2 : e_3] : t}$$

$$\frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, [t]) \\ F_d, S_d, R_s, E_s \vdash e_2 : t' \\ F_d, S_d, R_s, E_s \vdash e_2 : t'' \\ (t' = int \vee t' = \alpha) \wedge (t'' = int \vee t'' = \alpha) \\ t \text{ is unsafe}\end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2 : e_3] : t}\ \textbf{(TS-L-8a)}$$

Figure 28: Other List Operation Typing Rules

$$\textbf{(TS-L-8b)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : *ref(id, t) \\ F_d, S_d, R_s, E_s \vdash e_2 : t' \\ F_d, S_d, R_s, E_s \vdash e_2 : t'' \\ (t' = int \vee t' = \alpha) \wedge (t'' = int \vee t'' = \alpha) \\ t = \alpha \vee t = [t] \\ t = \alpha \vee t' = \alpha \vee t'' = \alpha\end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2 : e_3] : \alpha}$$

$$\frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : t \\ F_d, S_d, R_s, E_s \vdash e_2 : *ref(id, [t]) \\ t \text{ is safe}\end{array}}{F_d, S_d, R_s, E_s \vdash e_1\ in\ e_2 : bool}\ \textbf{(TS-L-9)}$$

$$\textbf{(TS-L-9a)}\ \frac{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1 : t \\ F_d, S_d, R_s, E_s \vdash e_2 : *ref(id, t') \\ t = \alpha \vee t = int \\ t' = \alpha \vee t' = [t''] \\ t = \alpha \vee t' = \alpha \vee (t' = [t''] \wedge t' \text{ is unsafe})\end{array}}{F_d, S_d, R_s, E_s \vdash e_1\ in\ e_2 : \alpha}$$

$$\frac{\begin{array}{c}id \text{ is a valid id} \\ F_d, S_d, R_s, E_s \vdash e : [t] \\ t \text{ is safe} \\ E'_s = newScope(E_s) \\ R'_s = declare(newScope(R_s), id, t) \\ F_d, S_d, R'_s, E'_s \vdash S\ : void \times t'\end{array}}{F_d, S_d, R_s, E_s \vdash for\ id\ in\ e\ \{S\} : void \times t'}\ \textbf{(TS-C-4)}$$

$$\textbf{(TS-C-4a)}\ \frac{\begin{array}{c}id \text{ is a valid id} \\ F_d, S_d, R_s, E_s \vdash e : [t] \\ E'_s = declare(newScope(E_s), id, t) \\ R'_s = newScope(R_s) \\ F_d, S_d, R'_s, E'_s \vdash S\ : t' \times t'' \\ t \text{ is unsafe} \vee t' = \alpha\end{array}}{F_d, S_d, R_s, E_s \vdash for\ id\ in\ e\ \{S\} : \alpha \times t''}$$

Figure 29: More Other List Operations

$$\frac{\begin{array}{c} id \text{ is a valid id} \\ F_d, S_d, R_s, E_s \vdash e : \alpha \\ E'_s = declare(newScope(E_s), id, \alpha) \\ R'_s = newScope(R_s) \\ F_d, S_d, R'_s, E'_s \vdash S \ : t' \times t'' \end{array}}{F_d, S_d, R_s, E_s \vdash for\ id\ in\ e\ \{S\} : \alpha \times t''} \textbf{ (TS-C-4b)}$$

Figure 30: More Other List Operations

4 Dynamic Semantics

4.1 Foreword

While this thesis focuses mainly on a type system, it's hard to talk about static semantics without also describing the dynamic semantics. I define two sets of semantics for this thesis: a Big-Step and a Small-Step operational semantics to rigorously describe the language and rigorously prove that the type system works. Most papers only do one. I chose big-step operational semantics because they rigorously describe the inner workings of the language while still being relatively easy to read (at least compared to other semantic approaches). Later, I describe a small-step operational semantics for a reduced set of the language so I can rigorously prove soundness and safety. There are a few important things to note. Firstly, I designed the semantics to handle programs that type-check, meaning programs deduced from the type rules. Secondly, I don't include errors in the Big-Step Semantics. Recall the system has safe and unsafe sides, and errors are a part of the unsafe side. Programs with errors in them cannot be derived. Errors are important for proofs but are otherwise uninteresting, so I only include errors in the Small-Step Semantics. Similarly to the typing rules, here is a list of the citations that influenced the semantic rules the most: 3, 4, 5, 8, 9, and 11.

4.2 Basics

Semantic rules are of the form: $F_d, S_d, H, H_m, R_s, E_s \vdash E \Downarrow H', H'_m, R'_s, E'_s, v, R_v$, meaning, in the context of function definitions F_d, structure definitions S_d, heap H, heap-mapping H_m, region set R_s, environment set E_s, the statement E, yields a new heap H', a new heap mapping H'_m, new region set R'_s, new environment set E'_s, a value v and return R_v. If a statement yields no value, $v = \_$ and if a statement yields no return $R_v = \_$.

4.3 Evaluation Context

$F_d : F_{id} \mapsto F_{id}(P_1 : T_2, P_2 : T_2, ...P_n : T_n) \rightarrow T\{S_1 S_2 ... S_m \ return \ e;\}$ (F_{id} is a function name, $P_1, P_2, ...P_n$ are parameters, $T_1, T_2, ...T_n$ are parameter types, $S_1 S_2 ... S_m \ return \ e$; are statements).

$S_d : S_{id} \mapsto S_{id}(A_1 : T_1, A_2 : T_2, ...A_n : T_n)$ (S_{id} is the structure name, $A_1, A_2, ...A_n$ are attribute names, $T_1, T_2, ...T_n$ are attribute types.

$H : l \mapsto v$ (l is a location and v a some value)

$H_m : a \mapsto l$ (l is a location and a is an address)

$R_s : id \mapsto a, l$ (l is a location and id is an identifier)

$E_s : id \mapsto a$ (id is an identifier and a is an address)

Figure 31: Evaluation Context

4.4 Addresses and Locations

Addresses and locations emulate memory addresses. I don't provide a precise definition because it's not important; they are devices to describe mappings and nothing more. However, I do precisely describe their use in the definitions and rules. Lastly, I use both addresses and locations because they make expressing references easier. This is also the purpose of the heap-mapping in the evaluation context.

4.5 Types of Values

nil evaluates to: nil

Integers evaluate to: $int\{i\}$, where i is some integer

Booleans evaluate to: $bool\{b\}$, where b is some boolean

References evaluate to: *ref{id, a} and ref{id, a}

Lists: $[a_1, a_2, ...a_n, \{l_1, l_2, ...l_m\}]$

Structures are of the form: $S_{id}(A_1 : a_1, A_2 : a_2, ...A_n : a_n, \{l_1, l_2, ...l_m\})$, $A_1, A_2, ...A_n$ are attribute names, $a_1, a_2, ...a_n$ are addressed where $a_i \in domain(H_m)$

Figure 32: Evaluation Context

4.6 Regions and Environments

Definition: closeScope $closeScope(R_s, E_s, H, H_m) = R'_s, E'_s, H', H'_m$

$R'_s = R_s$ without $R \in R_s$, where $R' \succeq R$, $\forall R' \in R_s$

$E'_s = E_s$ without $E \in E_s$, where $E' \succeq E$, $\forall E' \in E_s$

$H' = dealloc(H, H_m(id_1), H_m(id_2), ...H_m(id_n))$ where $id_1, id_2, ...id_n \in domain(R)$

$H'_m = H_m/id_1/id_2.../id_n/id'_1/id'_2.../id'_m$ where $id_1, id_2, ...id_n \in domain(R)$ and $id'_1, id'_2, ...id'_m \in domain(E)$

Definition: newScope

$newScope(Rs) = R'_s$ where $Rs' = R_s$ with some new Region $R = \emptyset$ such that $R' \succeq R, \forall R' \in R_s$

$newScope(Es) = E'_s$ where $Es' = E_s$ with some new Environment $E = \emptyset$ such that $E' \succeq E, \forall E' \in E_s$

Definition: declare

$declare(R_s, id, a, l) = R'_s$, $Rs' = R_s$ where $R[(a,l)/id]$ for $R = find(R_s, id)$

$declare(E_s, id, a) = E'_s$, $Es' = E_s$ where $E[a/id]$ for $E = find(E_s, id)$

Definition: find

$find(R_s, id) = R$, where $R' \succeq R$, $\forall R' \in R_s$ such that $R'(id) = (a,l)$ and $R(id) = (a,l)$

$find(E_s, id) = E$, where $E' \succeq E$, $\forall E' \in E_s$ such that $E'(id) = a$ and $R(id) = (a,l)$

Figure 33: Region and Environment Evaluation Functions

4.7 Memory

Definition: alloc $alloc(H, v) = H', l$, where $H' = H[v/l]$ and $l \notin domain(H)$
Definition: realloc $realloc(H, v) = H', l$, where $H(l) = v'$ and $H' = H[v''/l]$. The value of v'' depends on v and v':

If $v = nil, int\{i\}$ or $v = bool\{b\}$, then $v'' = v$.

If $v = S_{id}(...\{l_1, l_2, ...l_n\})$ and $v' = S_{id}(...\{l'_1, l'_2, ...l'_n\})$, then $v'' = S_{id}(...\{l_1, l_2, ...l_n, l'_1, l'_2, ...l'_n\}$.

If $v = List(...\{l_1, l_2, ...l_n\})$ and $v' = List(...\{l'_1, l'_2, ...l'_n\})$, then $v'' = List(...\{l_1, l_2, ...l_n, l'_1, l'_2, ...l'_n\})$.

Definition: dealloc $dealloc(H, l) = H'$, if $l \notin domain(H)$, then $H' = H$. If $H'(l) = v$, H' depends on the value of v :

If $v = int\{i\}$ or $v = bool\{b\}$, then $H' = H/l$

If $v = S_{id}(...\{l_1, l_2, ...l_n\})$, then $H'' = dealloc(H, l_1, l_2, ...l_n)$ and $H' = H''/l$.

If $v = List(...\{l_1, l_2, ...l_n\})$, then $H'' = dealloc(H, l_1, l_2, ...l_n)$ and $H' = H''/l$.

Figure 34: Memory Evaluation Functions

4.8 Other

Definition: $set(H_m, a, l) = H'_m \quad (H'_m = H_m[l/a])$

Definition: $newAddr(H_m, l) = H'_m, a \quad (H'_m = H_m[l/a])$ and $a \notin domain(H_m)$

Figure 35: Other Evaluation Functions

4.9 Abbreviations

$$
\begin{gathered}
alloc(H, v_1, v_2, ...v_n) = H', l_1, l_2, ...l_n, \text{ where:} \\
alloc(H, v_1) = H_1, l_1 \quad (H_1 = H[v_1/l_1]) \\
alloc(H_1, v_2) = H_2, l_2 \quad (H_2 = H_1[v_2/l_2]) \\
\vdots \\
alloc(H_{n-1}, v_n) = H', l_n \quad (H' = H_{n-1}[v_n/l_n]) \\
dealloc(H, l_1, l_2, ...l_n) = H', \text{ where:} \\
dealloc(H, l_1) = H_1 \\
dealloc(H_1, l_2) = H_2 \\
\vdots \\
dealloc(H_{n-1}, l_n) = H'
\end{gathered}
$$

$$
\begin{gathered}
newAddr(H_m, l_1, l_2, ...l_n) = H'_m, a_1, a_2, ...a_n, \text{ where:} \\
newAddr(H_m, l_1) = H_{m1}, a_1 \\
newAddr(H_{m1}, l_2) = H_{m2}, a_2 \\
\vdots \\
newAddr(H_{mn-1}, l_n) = H'_m, a_n
\end{gathered}
$$

$$
\begin{gathered}
declare(Rs, id_1, a_1, l_1, id_2, a_2, l_2, ...id_n, a_n, l_n) = Rs', \text{ where:} \\
declare(R_s, id1, a1, l1) = R_{s1} \\
declare(R_{s1}, id_2, a_2, l_2) = R_{s2} \\
\vdots \\
declare(R_{sn-1}, id_n, a_n, l_n) = R'_s
\end{gathered}
$$

$$
\begin{gathered}
declare(Es, id_1, a_1, id_2, a_2, ...id_n, a_n) = Es', \text{ where:} \\
declare(E_s, id1, a1, l1) = E_{s1} \\
declare(E_{s1}, id_2, a_2, l_2) = E_{s2} \\
\vdots \\
declare(E_{sn-1}, id_n, a_n, l_n) = E'_s
\end{gathered}
$$

$$
H_m(R_s(id)) = H_m(a) \text{ where } R_s(id) = (a, l)
$$

Figure 36: Abbreviations A

4.10 Semantic Rules

Simple Expressions

$$\textbf{(S-E-1)}\ \frac{\text{type is } int}{H, H_m, R_s, E_s, F_d, S_d \vdash i \Downarrow H, H_m, R_s, E_s, int\{i\}, \_}$$

$$\frac{\text{type is } bool}{F_d, S_d, H, H_m, R_s, E_s \vdash b \Downarrow H, H_m, R_s, E_s, bool\{b\}, \_}\ \textbf{(S-E-2)}$$

$$\frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow H', H'_m, R_s, E_s, bool\{b_1\}, \_ \\ b\ = bool\{\neg b_1\} \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash !e \Downarrow\ H', H'_m, R_s, E_s, bool\{b\}, \_}\ \textbf{(S-E-3)}$$

$$\textbf{(S-E-4)}\ \frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'', H''_m, R_s, E_s, int\{i_1\}, \_ \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash e_2 \Downarrow H', H'_m, R_s, E_s, int\{i_2\}, \_ \\ \text{op} \in \{+, -, *\} \\ i = i_1\ op\ i_2 \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1\ op\ e_2 \Downarrow\ H', H'_m, R_s, E_s, int\{i\}, \_}$$

$$\textbf{(S-E-5)}\ \frac{F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H', H'_m, R_s, E_s, bool\{false\}, \_}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1\ and\ e_2 \Downarrow H', H'_m, R_s, E_s, bool\{false\}, \_}$$

$$\frac{\begin{array}{c} F_d, S_d, H, R_s, E_s \vdash e_1 \Downarrow H'', H''_m, R_s, E_s, bool\{true\}, \_ \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash e_2 \Downarrow H', H'_m, R_s, E_s, bool\{b\}, \_ \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1\ and\ e_2 \Downarrow H', H'_m, R_s, E_s, bool\{b\}, \_}\ \textbf{(S-E-6)}$$

$$\textbf{(S-E-7)}\ \frac{F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H', H'_m, R_s, E_s, bool\{true\}, \_}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1\ or\ e_2 \Downarrow H', H'_m, R_s, E_s, bool\{true\}, \_}$$

$$\frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'', H''_m, R_s, E_s, bool\{false\}, \_ \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash e_2 \Downarrow H', H'_m, R_s, E_s, bool\{b\}, \_ \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1\ or\ e_2 \Downarrow H', H'_m, R_s, E_s, bool\{b\}, \_}\ \textbf{(S-E-8)}$$

Figure 37: Expression Evaluation Rules

More Simple Expressions

$$\textbf{(S-E-9)}\ \frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'', H''_m, R_s, E_s, int\{i_1\}, \_ \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash e_2 \Downarrow H', H'_m, R_s, E_s, int\{i_2\}, \_ \\ \text{op} \in \{<, >, ==\} \\ b = i_1\ op\ i_2 \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1\ op\ e_2 \Downarrow\ H', H'_m, R_s, E_s, bool\{b\}, \_}$$

Figure 38: More Expression Evaluation Rules

Variable Declarations

$$\textbf{(S-V-D-1)}\ \frac{\begin{array}{c}\text{type is } void \times void \\ F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow H'', H''_m, R_s, E_s, v, \_ \\ v \neq ref\{id', a'\} \\ alloc(H'', v) \ = \ H', l \\ newAddr(H''_m, l) = H'_m, a \\ declare(R_s, id, a, l) = R'_s\end{array}}{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash let\ id : t \ = \ e; \Downarrow H', H'_m, R'_s, E_s, \_, \_ \\ F_d, S_d, H, H_m, R_s, E_s \vdash let\ id \ = \ e; \Downarrow H', H'_m, R'_s, E_s, \_, \_\end{array}}$$

$$\textbf{(S-V-D-1a)}\ \frac{\begin{array}{c}\text{type is } \alpha \times void \\ F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow H'', H''_m, R_s, E_s, v, \_ \\ v \neq ref\{id', a'\} \\ alloc(H'', v) \ = \ H', l \\ newAddr(H''_m, l) = H'_m, a \\ declare(E_s, id, a) = E'_s\end{array}}{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash let\ id : t \ = \ e; \Downarrow H', H'_m, R_s, E'_s, \_, \_ \\ F_d, S_d, H, H_m, R_s, E_s \vdash let\ id \ = \ e; \Downarrow H', H'_m, R_s, E'_s, \_, \_\end{array}}$$

$$\frac{\begin{array}{c}\text{type is } void \times void \\ F_d, S_d, H, R_s, E_s \vdash e \Downarrow H', H''_m, R_s, E_s, v, \_ \\ v = ref\{id', a\} \\ newAddr(H''_m, H''_m(a)) = H'_m, a' \\ declare(R_s, id, a') = R'_s\end{array}}{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash let\ id : t = e; \Downarrow H', H'_m, R'_s, E_s, \_, \_ \\ F_d, S_d, H, H_m, R_s, E_s \vdash let\ id = e; \Downarrow H', H'_m, R'_s, E_s, \_, \_\end{array}}\ \textbf{(S-V-D-2)}$$

$$\frac{\begin{array}{c}\text{type is } \alpha \times void \\ F_d, S_d, H, R_s, E_s \vdash e \Downarrow H', R_s, E_s, v, \_ \\ v = ref\{id', a\} \\ newAddr(H''_m, H''_m(a)) = H'_m, a' \\ declare(E_s, id, a') = E'_s\end{array}}{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash let\ id : t = e; \Downarrow H', H'_m, R_s, E'_s, \_, \_ \\ F_d, S_d, H, H_m, R_s, E_s \vdash let\ id = e; \Downarrow H', H'_m, R_s, E'_s, \_, \_\end{array}}\ \textbf{(S-V-D-2a)}$$

Figure 39: Variable Declaration Evaluation Rules A

Variable Assignment Evaluation

$$\textbf{(S-V-A-1)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow H'', H'_m, R_s, E_s, v, \_ \\ v \neq ref\{id', a\} \\ find(R_s, id) = R \\ realloc(H'', H'_m(R(id)), v) \ = \ H'\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash id \ = \ e \ ; \Downarrow H', H'_m, R_s, E_s, \_, \_}$$

$$\textbf{(S-V-A-1a)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow H'', H'_m, R_s, E_s, v, \_ \\ v \neq ref\{id', a\} \\ find(E_s, id) = E \\ realloc(H'', H'_m(E(id)), v) \ = \ H'\end{array}}{F_d, S_d, H, H'_m R_s, E_s \vdash id \ = \ e \ ; \Downarrow H', H'_m, R_s, E_s, \_, \_}$$

$$\textbf{(T-V-A-2)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow H', H''_m, R_s, E_s, v, \_ \\ v = ref\{id', a\} \\ find(R_s, id) = R \\ set(H''_m, R(id), H''_m(a)) = H'_m\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash id = e; \Downarrow H, H'_m, R_s, E_s, \_, \_}$$

$$\frac{\begin{array}{c}F_d, S_d, H, R_s, E_s \vdash e \Downarrow H', H''_m, R_s, E_s, v, \_ \\ v = ref\{id', a\} \\ find(E_s, id) = E \\ set(H''_m, E(id), H''_m(a)) = H'_m\end{array}}{F_d, S_d, H, R_s, E_s \vdash id = e; \Downarrow H, H'_m R_s, E_s, \_, \_}\ \textbf{(T-V-A-2a)}$$

Figure 40: Variable Assignment Evaluation Rules

Variable Reference Evaluation

$$\textbf{(S-V-R-1)}\ \frac{find(R_s, id) = R \qquad v = H(H_m(R(id)))}{F_d, S_d, H, H_m, R_s, E_s \vdash id \Downarrow H, H_m, R_s, E_s, *ref\{id, v\}, \_}$$

$$\frac{find(E_s, id) = E \qquad v = H(H_m(E(id)))}{F_d, S_d, H, H_m, R_s, E_s \vdash id \Downarrow H, H_m, R_s, E_s, *ref\{id, v\}, \_}\ \textbf{(S-V-R-1a)}$$

$$\frac{}{F_d, S_d, H, H_m, R_s, E_s \vdash *ref\{id, v\} \Downarrow H, H_m, R_s, E_s, v, \_}\ \textbf{(S-V-R-3)}$$

$$\frac{find(R_s, id) = R \qquad R(id) = a}{F_d, S_d, H, H_m, R_s, E_s \vdash \&id \Downarrow H, H_m, R_s, E_s, ref\{id, a\}, \_}\ \textbf{(S-V-R-4)}$$

$$\frac{find(E_s, id) = E \qquad E(id) = a}{F_d, S_d, H, H_m, R_s, E_s \vdash \&id \Downarrow H, H_m, R_s, E_s, ref\{id, a\}, \_}\ \textbf{(S-V-R-4a)}$$

Figure 41: Variable Reference Evaluation Rules

Sequential Statements

$$\textbf{(S-S-1)}\ \frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash S_1 \Downarrow H'', H''_m, R''_s, E''_s, \_, \_ \\ F_d, S_d, H'', H''_m, R''_s, E''_s \vdash S_2 \Downarrow H', H'_m, R'_s, E'_s, \_, R_v \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash S_1 S_2 \Downarrow H', H'_m, R'_s, E'_s, \_, R_v}$$

$$\textbf{(S-S-2)}\ \frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash S_1 \Downarrow H', H'_m, R'_s, E'_s, \_, R_v \\ R_v \neq \_ \end{array}}{F_d, S_d, H, R_s, E_s \vdash S_1 S_2 \Downarrow H', H'_m, R'_s, E'_s, \_, R_v}$$

Figure 42: Sequential Statement Evaluation Rules

Control Flow

$$\textbf{(S-C-1a)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash b \Downarrow H''', H'''_m, R_s, E_s, bool\{true\}, \_ \\ newScope(R_s) = R''_s \qquad newScope(E_s) = E''_s \\ F_d, S_d, H''', H'''_m, R''_s, E''_s \vdash S_1 \Downarrow H'', H''_m, R'_s, E'_s, bool\{true\}, R_v \\ closeScope(R'_s, E'_s, H'', Hm'') = H', H'_m, R_s, E_s\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash if\ b\ \{S_1\}\ \{S_2\} \Downarrow H', H'_m, R_s, E_s, \_, R_v}$$

$$\textbf{(S-C-1b)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash b \Downarrow H''', H'''_m, R_s, E_s, bool\{false\}, \_ \\ newScope(R_s) = R''_s \qquad newScope(E_s) = E''_s \\ F_d, S_d, H''', H'''_m, R''_s, E''_s \vdash S_2 \Downarrow H'', H''_m, R'_s, E'_s, bool\{true\}, R_v \\ closeScope(R'_s, E'_s, H'', Hm'') = H', H'_m, R_s, E_s\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash if\ b\ \{S_1\}\ \{S_2\} \Downarrow H', H'_m, R_s, E_s, \_, R_v}$$

$$\textbf{(S-C-2a)}\ \frac{F_d, S_d, H, H_m, R_s, E_s \vdash b \Downarrow H', H'_m, R_s, E_s, bool\{false\}, \_}{F_d, S_d, H, H_m, R_s, E_s \vdash while\ b\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, \_}$$

$$\textbf{(S-C-2b)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash b \Downarrow H'''', H''''_m, R_s, E_s, bool\{true\}, \_ \\ newScope(R_s) = R''_s \qquad newScope(E_s) = E''_s \\ F_d, S_d, H'''', H''''_m, R''_s, E''_s \vdash S \Downarrow H''', H'''_m, R'_s, E'_s, \_, \_ \\ closeScope(R'_s, E'_s, H''', Hm''') = H'', H''_m, R_s, E_s \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash while\ b\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, R_v\end{array}}{F_d, S_d, H, R_s, E_s \vdash while\ b\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, R_v}$$

$$\textbf{(S-C-2c)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash b \Downarrow H''', H'''_m, R_s, E_s, bool\{true\}, \_ \\ newScope(R_s) = R''_s \qquad newScope(E_s) = E''_s \\ F_d, S_d, H''', H'''_m, R''_s, E''_s \vdash S \Downarrow H'', H''_m, R'_s, E'_s, \_, R_v \\ closeScope(R'_s, E'_s, H'', Hm'') = H', H'_m, R_s, E_s\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash while\ b\ \{S\} \Downarrow H', H'_m, R'_s, E'_s, \_, R_v}$$

Figure 43: Control Flow Evaluation Rules

Return

$$\textbf{(S-R-1)}\ \frac{F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow H', H'_m, R_s, E_s, v, \_}{F_d, S_d, H, H_m, R_s, E_s \vdash return\ e; \Downarrow H', H'_m, R_s, E_s, \_, v}$$

Figure 44: Return Evaluation Rule

Function Calls

$$\textbf{(S-F-1)}\ \frac{\begin{array}{c} type\ of:\ e_{a1}, e_{a2}, ...\ e_{ab}\ is\ not\ \alpha \\ type\ of:\ e_{c1}, e_{c2}, ...\ e_{cd}\ is\ \alpha \\ F_d(F_{id}) = F_{id}(P_1 : T_1, P_2 : T_2, ..., P_n : T_n) \rightarrow R_t\{S_1 S_2\ ...\ S_j\ return\ e;\} \\ F_d, S_d, H, H_m, R_s, E_s \vdash e_{a1} \Downarrow H_1, H_{m1}, R_s, E_s, v_1, \_ \\ F_d, S_d, H_1, H_{m1}, R_s, E_s \vdash e_{a2} \Downarrow H_2, H_{m2}, R_s, E_s, v_2, \_ \\ \vdots \\ F_d, S_d, H_{b-1}, H_{mb-1}, R_s, E_s \vdash e_{ab} \Downarrow H_b, H_{mb}, R_s, E_s, v_b, \_ \\ F_d, S_d, H_b, H_{mb}, R_s, E_s \vdash e_{c1} \Downarrow H_{b+1}, H_{mb+1}, R_s, E_s, v_{b+1}, \_ \\ F_d, S_d, H_{b+1}, H_{mb+1}, R_s, E_s \vdash e_{c2} \Downarrow H_{b+2}, H_{mb+2}, R_s, E_s, v_{b+2}, \_ \\ \vdots \\ F_d, S_d, H_{n-1}, H_{mn-1}, R_s, E_s \vdash e_{cd} \Downarrow H_n, H_{mn}, R_s, E_s, v_n, \_ \\ alloc(H_n, v_1, v_2, ...v_b, v_{b+1}, v_b{+}2, ...v_n) = H'', l_1, l_2, ...\ l_b, l_{b+1}, l_{b+2}, ...\ l_n \\ newAddr(H_{mn}, l_1, l_2, ...l_b, l_{b+1}, l_{b+2}, ...\ l_n) = H''_m, a_1, a_2, ...\ a_b, a_{b+1}, a_{b+2}, ...\ a_n \\ R'_s = \emptyset \qquad E'_s = \emptyset \\ declare(R'_s, P_1, a_1, l_1, P_2, a_2, l_2, ...P_b, a_b, l_b) = R''_s \\ declare(E'_s, P_{b+1}, a_{b+1}, P_{b+2}, a_{b+2}, ...P_n, a_n, l_n) = E''_s \\ F_d, S_d, H_n, H_{mn}, R''_s, E''_s \vdash S_1 S_2\ ...\ S_j\ return\ e; \Downarrow H'', H''_m, R''_s, E''_s, \_, R_v \\ closeScope(R''_s, E''_s, H'', H''_m) = R'_s, E'_s, H', H'_m \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash F_{id}(e_{a1}, e_{a2}, ...\ e_{ab}, e_{c1}, e_{c2}, ...\ e_{cd}) \Downarrow H', H'_m, R_s, E_s, R_v, \_}$$

Figure 45: Function Call Evaluation Rules

Structure Literal

$$\textbf{(TS-S-L-1)}\ \frac{\begin{array}{c} S_d(S_{id}) = S_{id}(S_{id}) = S_{id}(A_1 : T_1,\ A_2 : T_2, ...,\ A_n : T_n) \\ F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H_1, H_{m1}, R_s, E_s, v_1, \_ \\ F_d, S_d, H_1, H_{m1}, R_s, E_s \vdash e_2 \Downarrow H_2, H_{m2}, R_s, E_s, v_2, \_ \\ \vdots \\ F_d, S_d, H_{n-1}, H_{mn-1}, R_s, E_s \vdash e_n \Downarrow H_n, H_{mn}, R_s, E_s, v_n, \_ \\ alloc(H_n, v_1, v_2, ...v_n) = H', l_1, l_2, ...\ l_n \\ newAddr(H_{mn}, l_1, l_2, ...\ l_n) = H'_m, a_1, a_2, ...\ a_n \end{array}}{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash S_{id}\{A_1 = e_1,\ A_2 = e_2, ...,\ A_n : e_n\} \Downarrow \\ H', H'_m, R_s, E_s, S_{id}(A_1 : a_1, A_2 : a_2, ...A_n : a_n, \{l_1, l_2, ...\ l_n\}), \_ \end{array}}$$

Figure 46: Structure Literal Evaluation Rule

Structure Assignment

$$\textbf{(T-S-A-1)}\ \frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow \\ H''', H''_m, R_s, E_s, S_{id}(A_1 : a_1, A_2 : a_2, ...A_n : a_n, \{l_1, l_2, ...l_n\}), \_ \\ id = A_i \in A_1, A_2, ...A_n \\ F_d, S_d, H''', H''_m, R_s, E_s \vdash e_2 \Downarrow H'', H'_m, R_s, E_s, v, \_ \\ realloc(H'', H'_m(a_i), v) = H' \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1.id = e_2 \Downarrow H', H'_m, R_s, E_s, \_, \_}$$

$$\frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow \\ H'', H'''_m, R_s, E_s, S_{id}(A_1 : a_1, A_2 : a_2, ...A_n : a_n, \{l_1, l_2, ...l_n\}), \_ \\ id = A_i \in A_1, A_2, ...A_n \\ F_d, S_d, H'', H'''_m, R_s, E_s \vdash e_2 \Downarrow H', H''_m, R_s, E_s, ref\{id', a\}, \_ \\ set(H''_m, a_i, H''_m(a))) = H'_m \end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1.id = e_2 \Downarrow H', H'_m, R_s, E_s, \_, \_}\ \textbf{(T-S-A-2)}$$

Figure 47: Structure Assignment Evaluation Rules A

Structure References

$$\textbf{(T-S-R-1)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow \\ H', H'_m, R_s, E_s, *ref\{id', S_{id}(A_1 : a_1, A_2 : a_2, ...A_n : a_n, \{l_1, l_2, ...l_n\})\}, \_ \\ id = A_i \in A_1, A_2, ...A_n \qquad v = *ref\{id', H(H_m(a_i))\}\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e.id \Downarrow H', H'_m, R_s, E_s, v, \_}$$

$$\frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e \Downarrow \\ H', H'_m, R_s, E_s, *ref\{id', S_{id}(A_1 : a_1, A_2 : a_2, ...A_n : a_n, \{l_1, l_2, ...l_n\})\}, \_ \\ id = A_i \in A_1, A_2, ...A_n\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash \&e.id \Downarrow H', H'_m, R_s, E_s, ref\{id', a_i\}, \_}\ \textbf{(T-S-R-2)}$$

Figure 48: Structure Reference Evaluation Rules A

Global Definitions

$$\textbf{(TS-G-1)}\ \frac{\begin{array}{c}V'_1 = F_{id1}(P_{11} : T_{11},\ P_{12} : T_{12}, ...,\ P_{1n_1} : T_{1n_1}) \to T_{r1}\{S_{11}S_{12}...S_{1x_1}\ return\ e_1;\} \\ V'_2 = F_{id2}(P_{21} : T_{21},\ P_{22} : T_{22}, ...,\ P_{2n_2} : T_{2n_2}) \to T_{r2}\{S_{21}S_{22}...S_{2x_1}\ return\ e_2;\} \\ \vdots \\ V'_j = F_{idj}(P_{j1} : T_{j1},\ P_{j2} : T_{j2}, ...,\ P_{jn_j} : T_{j_{n_j}}) \to T_{rj}\{S_{j1}S_{j2}...S_{jx_j}\ return\ e_j;\} \\ V'_{j+1} = S_{id1}(A_{11} : T''_{11},\ A_{12} : T''_{12}, ...,\ A_{1m_1} : T''_{1m_1}) \\ V'_{j+2} = S_{id2}(A_{21} : T''_{21},\ A_{22} : T''_{22}, ...,\ A_{2m_2} : T''_{2m_2}) \\ \vdots \\ V'_{j+k} = S_{idk}(A_{k1} : T''_{k1},\ A_{k2} : T''_{k2}, ...,\ A_{km_k} : T''_{km_k}) \\ F_{d1} = F_d[V'_1\ /F_{id1}][V'_2\ /F_{id2}]\ ...,\ [V'_j\ /F_{idj}] \\ S_{d1} = S_d[V'_{j+1}\ /S_{id1}][V'_{j+2}\ /S_{id2}]\ ...,\ [V'_{j+k}\ /S_{idk}] \\ F_{d1}, S_{d1}, H, H_m, R_s, E_s \vdash def\ F_{id1}(P_{11} : T_{11},\ P_{12} : T_{12}, ...,\ P_{1n_1} : T_{1n_1}) \to T_{r1}\ \{S_{11}S_{12}...S_{1x_1}\ return\ e_1;\} : \\ H', H'_m, R_s, E_s, V'_1, \_ \\ F_{d1}, S_{d1}, H, H_m, R_s, E_s \vdash def\ F_{id2}(P_{21} : T_{21},\ P_{22} : T_{22}, ...,\ P_{2n_2} : T_{2n_2}) \to T_{r2}\ \{S_{21}S_{22}...S_{2x_2}\ \ return\ e_2;\} : \\ H', H'_m, R_s, E_s, V'_2, \_ \\ \vdots \\ F_{d1}, S_{d1}, H, H_m, R_s, E_s \vdash def\ F_{idj}(P_{j1} : T_{j1},\ P_{j2} : T_{j2}, ...,\ P_{j_{n_j}} : T_{j_{n_j}}) \to T_{rj}\ \{S_{j1}S_{j2}...S_{jx_j}\ \ return\ e_j;\} : \\ H', H'_m, R_s, E_s, V'_j, \_ \\ F_{d1}, S_{d1}, H, H_m, R_s, E_s \vdash struct\ S_{id1}\{A_{11} : T''_{11},\ A_{12} : T''_{12}, ...,\ A_{1m_1} : T''_{1m_1}\} : \\ H', H'_m, R_s, E_s, V'_{j+1}, \_ \\ F_{d1}, S_{d1}, H, H_m, R_s, E_s \vdash struct\ S_{id2}\{A_{21} : T''_{21},\ A_{22} : T''_{22}, ...,\ A_{2m_2} : T''_{2m_2}\} : \\ H', H'_m, R_s, E_s, V'_{j+2}, \_ \\ \vdots \\ F_{d1}, S_{d1}, H, H_m, R_s, E_s \vdash struct\ S_{idk}\{A_{k1} : T''_{k1},\ A_{k2} : T''_{k2}, ...,\ A_{km_k} : T''_{km_k}\}\ S : \\ H', H'_m, R_s, E_s, V'_{j+k}, \_ \\ F_{d1}, S_{d1}, H, H_m, R_s, E_s \vdash\ S \Downarrow H', H'_m, R'_s, E'_s, \_, R_v \\ 0 \le j, k \\ 0 \le n_1, n_2, ...n_j \\ 0 \le x_1, x_2, x_3, ..., x_j \\ 0 \le m_1, m_2, ...m_k \\ S_{id}\text{ are unique} \qquad F_{id}\text{ are unique} \\ A_i,\ P_i\text{ are unique in the context of their definitions}\end{array}}{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash def\ F_{id1}(P_{11} : T_{11},\ P_{12} : T_{12}, ...,\ P_{1n_1} : T_{1n_1}) \to T_{r1}\ \{S_{11}S_{12}...S_{1x_1}\ return\ e_1;\} \\ def\ F_{id2}(P_{21} : T_{21},\ P_{22} : T_{22}, ...,\ P_{2n_2} : T_{2n_2}) \to T_{r2}\ \{S_{21}S_{22}...S_{2x_1}\ return\ e_2;\} \\ \vdots \\ def\ F_{idj}(P_{j1} : T_{j1},\ P_{j2} : T_{j2}, ...,\ P_{jn_j} T_{j_{n_j}}) \to T_{rj}\ \{S_{j1}S_{j2}...S_{jx_j}\ return\ e_j;\} \\ struct\ S_{id1}\{A_{11} : T''_{11},\ A_{12} : T''_{12}, ...,\ A_{1m_1} : T''_{1m_1}\} \\ struct\ S_{id2}\{A_{21} : T''_{21},\ A_{22} : T''_{22}, ...,\ A_{2m_2} : T''_{2m_2}\} \\ \vdots \\ struct\ S_{idk}\{A_{k1} : T''_{k1},\ A_{k2} : T''_{k2}, ...,\ A_{km_k} : T''_{km_k}\}\ S \Downarrow H', H'_m, R'_s, E'_s, \_, R_v\end{array}}$$

Figure 49: Global Definitions Evaluation Rules A

List Literals

$$\textbf{(TS-L-1)}\ \frac{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H_1, H_{m1}, R_s, E_s, v_1, \_ \\ F_d, S_d, H_1, H_{m1}, R_s, E_s \vdash e_2 \Downarrow H_2, H_{m2}, R_s, E_s, v_2, \_ \\ \vdots \\ F_d, S_d, H_{n-1}, H_{mn-1}, R_s, E_s \vdash e_n \Downarrow H_n, H_{mn}, R_s, E_s, v_n, \_ \\ \forall v \in v_1, v_2, ...v_n : v \neq ref(id, a) \\ alloc(H_n, v_1, v_2, ...v_n) = H', l_1, l_2, ...l_n \\ newAddr(H_{mn}, l_1, l_2, ...l_n) = H'_m, a_1, a_2, ...a_n \end{array}}{\begin{array}{c} F_d, S_d, H, H_m, R_s, E_s \vdash [e_1, e_2, ...e_n] \Downarrow \\ H', H'_m, R_s, E_s, List(a_1, a_2, ...a_n, \{l_1, l_2, ...l_n\}), \_ \end{array}}$$

Figure 50: List Literal Evaluation Rule

List Operations

$$\textbf{(TS-L-O-1)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'''', H'''_m, R_s, E_s, *ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 0 \\ F_d, S_d, H'''', H'''_m, R_s, E_s \vdash e_2 \Downarrow H''', H''_m, R_s, E_s, v_2, \_ \qquad v_2 \neq ref\{id', a'\} \\ H'', l = alloc(H''', v_2) \qquad H'_m, a_{n+1} = newAddr(H''_m, l) \\ H' = realloc(H'', H'_m(E(id)), List(a_1, a_2, \ldots\ a_n, a_{n+1}, \{l_1, l_2, \ldots l_n, l\}))\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e_1 << e_2; \Downarrow H', H'_m, R_s, E_s, \_, \_}$$

$$\textbf{(TS-L-O-2)}\ \frac{\begin{array}{c}H, H_m, R_s, E_s, F_d, S_d \vdash e \Downarrow H'', H'_m, R_s, E_s, *ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 1 \qquad v = H''(H'_m(a_n)) \\ v_r = List(a_1, a_2, \ldots\ a_{n-1}, \{l_1, l_2, \ldots l_{n-1}\}) \qquad H' = realloc(H'', H'_m(a), v_r)\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash e! \Downarrow H', H'_m, R_s, E_s, v, \_}$$

$$\textbf{(TS-L-3)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'', H''_m, R_s, E_s, *ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 1 \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash e_2 \Downarrow H', H'_m, R_s, E_s, int\{i\}, \_ \\ v = H'(H'_m(a_i)) \qquad n \geq i\end{array}}{\begin{array}{c}F_d, S_d, R_s, E_s \vdash e_1[e_2] \Downarrow H', H'_m, R_s, E_s, *ref\{id, v\}, \_ \\ F_d, S_d, R_s, E_s \vdash \&e_1[e_2] \Downarrow H', H'_m, R_s, E_s, ref\{id, a\}, \_\end{array}}$$

$$\textbf{(TS-L-4)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'''', H'''_m, R_s, E_s, *ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 1 \\ F_d, S_d, H'''', H'''_m, R_s, E_s \vdash e_2 \Downarrow H''', H''_m, R_s, E_s, int\{i\}, \_ \\ F_d, S_d, H''', H''_m, R_s, E_s \vdash e_3 \Downarrow H'', H'_m, R_s, E_s, v, \_ \\ v \neq ref\{id', a'\} \qquad n \geq i \qquad H' = realloc(H'', H'_m(a_i), v)\end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2] = e_3; \Downarrow H', H'_m, R_s, E_s, \_, \_}$$

$$\textbf{(TS-L-4a)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H''', H'''', R_s, E_s, *ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 1 \\ F_d, S_d, H''', H'''', R_s, E_s \vdash e_2 \Downarrow H'', H'''_m, R_s, E_s, int\{i\}, \_ \\ F_d, S_d, H'', H'''_m, R_s, E_s \vdash e_3 \Downarrow H', H''_m, R_s, E_s, *ref\{id', a'\}, \_ \\ n \geq i \qquad H'_m = set(H''_m, a, H''_m(a'))\end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2] = e_3; \Downarrow H', H'_m, R_s, E_s, \_, \_}$$

$$\textbf{(TS-L-5)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'', H''_m, R_s, E_s, *ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 0 \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash e_2 \Downarrow H', H'_m, R_s, E_s, *ref\{id', lst'\}, \_ \\ lst' = List(a'_1, a'_2, \ldots\ a'_m, \{l'_1, l'_2, \ldots l'_m\}) \qquad m \geq 0 \\ v = List(a_1, a_2, \ldots a_n, a'_1, a'_2, \ldots a'_m, \{l_1, l_2, \ldots l_n, l'_1, l'_2, \ldots l'_m\})\end{array}}{F_d, S_d, R_s, E_s \vdash e_1 + e_2 \Downarrow H', H'_m, R_s, E_s, v, \_}$$

Figure 51: List Operation Evaluation Rules

List Operations

$$\textbf{(TS-L-6)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H''', H'''_m, R_s, E_s, *ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 1 \\ F_d, S_d, H''', H'''_m, R_s, E_s \vdash e_2 \Downarrow H'', H''_m, R_s, E_s, int\{i_1\}, \_ \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash e_3 \Downarrow H', H'_m, R_s, E_s, int\{i_2\}, \_ \\ i_1 \leq i_2 \leq n \\ v = List(a_{i1}, a_{i1+2}, \ldots a_{i2}, \{l_{i1}, l_{i1+2}, \ldots l_{i2}\})\end{array}}{F_d, S_d, R_s, E_s \vdash e_1[e_2 : e_3] \Downarrow H', H'_m, R_s, E_s, v, \_}$$

$$\textbf{(TS-L-7a)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'', H''_m, R_s, E_s, v, \_ \\ F_d, S_d, H, H_m, R_s, E_s \vdash e_2 \Downarrow H', H'_m, R_s, E_s, ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 1 \\ \exists a_i \in a_1, a_2, \ldots\ a_n : H'(H'_m(a_i)) = v\end{array}}{F_d, S_d, R_s, E_s \vdash e_1\ in\ e_2 \Downarrow H', H'_m, R_s, E_s, bool\{true\}, \_}$$

$$\textbf{(TS-L-7b)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'', H''_m, R_s, E_s, v, \_ \\ F_d, S_d, H, H_m, R_s, E_s \vdash e_2 \Downarrow H', H'_m, R_s, E_s, ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots\ a_n, \{l_1, l_2, \ldots l_n\}) \qquad n \geq 0 \\ \nexists a_i \in a_1, a_2, \ldots\ a_n : H'(H'_m(a_i)) = v\end{array}}{F_d, S_d, R_s, E_s \vdash e_1\ in\ e_2 \Downarrow H', H'_m, R_s, E_s, bool\{false\}, \_}$$

$$\textbf{(TS-L-8)}\ \frac{\begin{array}{c}F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H', H'_m, R_s, E_s, ref\{id, lst\}, \_ \\ lst = List(a_1, a_2, \ldots a_n, \{l_1, l_2, \ldots l_n\}) \qquad n = 0\end{array}}{F_d, S_d, R_s, E_s \vdash for\ id\ in\ e\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, \_}$$

$$\textbf{(TS-L-9a)}\ \frac{\begin{array}{c}\text{type is } \alpha \times t \\ F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H''', H'''_m, R_s, E_s, ref\{id', lst\}, \_ \\ lst = List(a_1, a_2, \ldots a_n, \{l_1, l_2, \ldots l_n\}) \qquad n > 0 \\ R'_s = newScope(R_s) \qquad E'_s = newScope(E_s) \qquad E''_s = declare(id, a_1) \\ H''', H'''_m, R'_s, E''_s, F_d, S_d \vdash S \Downarrow H'', H''_m, R'_s, E''_s, \_, R_v \\ closeScope(R'_s, E''_s, H'', H''_m) = R_s, E_s, H', H'_m\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash for\ id\ in\ e\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, R_v}$$

$$\textbf{(TS-L-9b)}\ \frac{\begin{array}{c}\text{type is } \alpha \times t \\ F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'''', H''''_m, R_s, E_s, ref\{id', lst\}, \_ \\ lst = List(a_1, a_2, \ldots a_n, \{l_1, l_2, \ldots l_n\}) \qquad n > 0 \\ R'_s = newScope(R_s) \qquad E'_s = newScope(E_s) \qquad E''_s = declare(id, a_1) \\ H'''', H''''_m, R'_s, E''_s, F_d, S_d \vdash S \Downarrow H''', H'''_m, R'_s, E''_s, \_, \_ \\ closeScope(R'_s, E''_s, H''', H'''_m) = R_s, E_s, H'', H''_m \\ H'', H''_m, R_s, E_s, F_d, S_d \vdash for\ id\ in\ e[1:n]\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, R_v\end{array}}{H, H_m, R_s, E_s, F_d, S_d \vdash for\ id\ in\ e\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, R_v}$$

Figure 52: More List Operation Evaluation Rules

List Operations

$$\textbf{(TS-L-9c)}\ \frac{\begin{array}{c}\text{type is } void \times t \\ F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H''', H'''_m, R_s, E_s, ref\{id', lst\}, \_ \\ lst = List(a_1, a_2, ...a_n, \{l_1, l_2, ...l_n\}) \qquad n > 0 \\ R'_s = newScope(R_s) \qquad E'_s = newScope(E_s) \qquad R''_s = declare(id, a_1, l_1) \\ F_d, S_d, H''', H'''_m, R''_s, E'_s \vdash S \Downarrow H'', H''_m, R''_s, E'_s, \_, R_v \\ closeScope(R''_s, E'_s, H'', H''_m) = R_s, E_s, H', H'_m\end{array}}{H, H_m, R_s, E_s, F_d, S_d \vdash for\ id\ in\ e\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, R_v}$$

$$\textbf{(TS-L-6a)}\ \frac{\begin{array}{c}\text{type is } \alpha \times t \\ F_d, S_d, H, H_m, R_s, E_s \vdash e_1 \Downarrow H'''', H''''_m, R_s, E_s, ref\{id', lst\}, \_ \\ lst = List(a_1, a_2, ...a_n, \{l_1, l_2, ...l_n\}) \qquad n > 0 \\ R'_s = newScope(R_s) \qquad E'_s = newScope(E_s) \qquad R''_s = declare(id, a_1, l_1) \\ H'''', H''''_m, R''_s, E'_s, F_d, S_d \vdash S \Downarrow H''', H'''_m, R''_s, E'_s, \_, \_ \\ closeScope(R''_s, E'_s, H''', H'''_m) = R_s, E_s, H'', H''_m \\ F_d, S_d, H'', H''_m, R_s, E_s \vdash for\ id\ in\ e[1:n]\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, R_v\end{array}}{F_d, S_d, H, H_m, R_s, E_s \vdash for\ id\ in\ e\ \{S\} \Downarrow H', H'_m, R_s, E_s, \_, R_v}$$

Figure 53: Even More List Operation Evaluation Rules

5 Reference System

I provide an overview of lifetimes and regions through the reference system. I also show the reference system because, while it is simple, it voids restrictions imposed by traditional Region-Based Memory Systems.

```
1  struct someStruct {someAtt:int}
2
3  let someVariable = 10;
4  let str = someStruct{someAtt=someVariable};
5  let lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, someVariable];
6
7  # someVariable == str.someAtt == lst[10] == 10
8
9  someVariable = 11;
10
11 # str.someAtt == lst[10] == 10
```

Figure 54: Reference Example 1

In Figure 54, value references work like in most other languages. When someVariable was referenced, the Heap made a copy independent from the instance stored for someVariable. This isn't out of the ordinary. These types of references, while useful are limited. So, there is another type of reference, address references.

```
struct someStruct {someAtt:int}

let someVariable = 10;
let str = someStruct{someAtt=someVariable};
str.someAtt = &someVariable;
let lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, someVariable];
lst[10] = &someVariable;

# someVariable == str.someAtt == lst[10] == 10

someVariable = 11;

# someVariable == str.someAtt == lst[10] == 11
```

Figure 55: Reference Example 2

Figure 55 uses address references. Since str.someAtt, lst[10] addresses both map to the same location on the heap as someVariable, when someVariable changes, str.someAtt and lst[10] also change. Address references are simple but robust. Memory is allocated to the heap automatically; manual allocation isn't necessary. Also, there isn't a special type annotation or variable to hold addresses like in C. Since there is no special syntax for storing addresses, there is no need for a special syntax for dereferencing. However, there is a special syntax for passing an address reference to distinguish them from value-style references.

Despite the simplicity, this reference system isn't lacking in ability. You can create and assign references to variables, list indices, function parameters, and structure attributes. See Figure 56. The language emulates pointer arithmetic using lists. Addresses are added and removed from lists using primitive operators, and list indices are mutable, referencable, and accessible. Lastly, lists are nestable and can hold references.

```
def someFunction(i:int) -> int {return 1 + i;}
struct someStruct {someAtt:int}

let someVariable = 10;
let str = someStruct{someAtt=someVariable};
let lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, someVariable];

str.someAtt = &lst[10];
someVariable = &str.someAtt;

lst[10] = someFunction(&lst[10]);
# lst[10] == 11

let unsafeVariable: alpha = [1, true, 2, false];

unsafeVariable!;
# unsafeVariable == [1, true, 2]
unsafeVariable!;
# unsafeVariable == [1, true]

unsafeVariable << someStruct{someAtt = 6};
# unsafeVariable == [1, true, someStruct{someAtt=6}]
```

Figure 56: Reference Example 3

The type system guarantees that the safe side of the language cannot have memory errors. Thus, the type system ensures that only safe reference assignments are valid. It does this through lifetimes. For the safe side of the language, lifetimes depend mostly on variables. The lifetime of a variable spans from the time it's created to when it's destroyed. Regions track the lifetimes of variables in the safe side of the language and govern their use. Each program has a set of Regions. You can add new mappings to Regions, but when a Region is closed, it deallocates every mapping. Regions are closed automatically and only when it's safe, mainly by using lifetimes. Generally speaking, if something exists in some region R, it can only access and be bound to things of the same type that outlive it. The type rules ensure that programs that satisfy this type check.

$$\textbf{(TS-R-1)}\ \frac{\begin{array}{c} find(R_s, id) = R \wedge R(id) \ = \ t \ \vee \\ find(E_s, id) = E \wedge E(id) = t \end{array}}{\begin{array}{c} F_d, S_d, R_s, E_s \vdash id \ : *ref(id,t) \\ F_d, S_d, R_s, E_s \vdash \&id \ : ref(id,t) \\ F_d, S_d, R_s, E_s \vdash ref\{id, a\} \ : ref(id,t) \end{array}} \quad \frac{}{F_d, S_d, R_s, E_s \vdash id \ : *ref(id,t) : t}\ \textbf{(TS-R-2)}$$

Figure 57: Simple Variable Typing Rules With Scope

Figure 57 gives the typing judgements for both kinds of references. The first type of reference is for value style references. If id is valid, it will first evaluated to *ref(id, t), by **(TS-R-2)** it can be reduced to t using the subtyping rules. I use an intermediate form because it is helpful to know the variable being referenced and not just the type, especially when accessing lists and structures. Recall, Regions manage lifetimes of variables, so it's important to know which variable is being referenced and not just its type.

The other type of references are address style references. In Figure 5, &id evaluates to ref{id, a}, an intermediate form in the semantics, for the same purpose as the other style of reference in the type system. The type of &id and ref{id, a} is ref(id, t). Both types of reference encode the type of the variable and the lifetime by including id. When and where these are allowed to be assigned depends on the assigner, not the

assignee; to see the criteria, see the rules for assignment.

The only safety check reference rules perform is to ensure that the id is in scope. The reference typing rules ensure that the variable exists in a valid Region or an Environment. I only list the typing rules for references to safe variables, but I list all the rules in the type system.

$$\textbf{(S-V-R-1)}\ \frac{find(R_s, id) = R \qquad v = H(H_m(R(id)))}{F_d, S_d, H, H_m, R_s, E_s \vdash id \Downarrow H, H_m, R_s, E_s, *ref\{id, v\}, \_}$$

$$\frac{}{F_d, S_d, H, H_m, R_s, E_s \vdash *ref\{id, v\} \Downarrow H, H_m, R_s, E_s, v, \_}\ \textbf{(S-V-R-3)}$$

$$\frac{find(R_s, id) = R \qquad R(id) = a}{F_d, S_d, H, H_m, R_s, E_s \vdash \&id \Downarrow H, H_m, R_s, E_s, ref\{id, a\}, \_}\ \textbf{(S-V-R-4)}$$

Figure 58: Reference Typing Rules

Figure 58 presents the semantic rules for both kinds of safe references. Since the type system ensures that variables will exist in a Region and on the Heap, the premise of the first rule is just to *find* the value; it isn't a check, whereas, for the unsafe side of the semantics, the value or address might not exist, thus, it also serves as a check.

Lastly, the purpose of the Heap Map is to provide a layer of indirection between the Heap, Regions, and Environments. Recall Regions and Environments map identifiers to addresses, and Heap Maps map addresses to locations. Thus, you can change what an identifiers map to on the heap without changing Regions or Environments. This is useful for references of both kinds.

6 Soundness

6.1 Foreword

Soundness is arguably the most important proof for type systems. In essence, soundness proves a type system works. Soundness states that if a type system yields type T, for some program P, P is either a primitive value, or P evaluates to P', and the type system yields type T for P'. In short, it shows showing the type is static throughout the execution of a program, thus ensuring that the type system works and the guarantees hold.

Traditionally, a soundness argument has two parts: progress and preservation. Progress states that in the empty typing context, every statement is primitive or reduces to something. Preservation states that if, in some typing context, a statement is a certain type and it reduces, the reduced version will be the same type. I prove progress and preservation for the safe side and the unsafe side. Furthermore, I split the inductive assumption into two parts: the assumption for expressions typed t and statements typed $T_1 \times T_2$. Technically, it is one inductive assumption, but splitting it into two parts highlights what I'm trying to prove.

I also present a reduced, more complete system so that soundness is easier to prove. Most papers do this. It follows the same core idea of extending gradual types to regions, so I conjecture the first system gives the same guarantee of memory safety. Also, in a very hand-wavey way, I've spent a lot of time working with the former system, and I'm confident in its capabilities. Moreover, if you incorporate the type system in a compiler, the first type system presents the idea in a much more useful way; this system makes rigorous proofs easier. The semantics are also different. Before, I used intuitive and readable "Big-Step Semantics" to rigorously describe the language. However, for certain induction proofs, Big-Step semantics don't work well. So, I use what is called "Small-Step Semantics". In Small-Step Semantics, you have to specify rules as simple reductions. A Small Step Semantics system reduces statements one step at a time, making them harder

to write and read. However, certain induction proofs are now easier.

Lastly, this is far and away the most popular approach to proving soundness. The proof is my own, but broadly speaking, the structure and approach to semantics are not. These references influenced these proofs: 3, 4, 5, 6, 7, 8 and 9.

Typing Context

$R : id \mapsto t$ (id is an identifier and t is a type)

$E : id \mapsto t$ (id is an identifier and t is a type)

Evaluation Context

$V : id \mapsto l$ (id is an identifier and l is a location)

$H : l \mapsto v$ (l is a location and v is a value)

Figure 59: Evaluation and Typing Contexts

Types

Integers: int

Boolean: bool

Unsafe, Dynamic Type: α

Value Style References: *ref(id, t)

Address Style References: ref(id, t)

Structure Types: $S_{id}(A_1 : T_1, A_2 : T_2, ...A_n : T_n)$ (S_{id} is the structure name, $A_1, A_2, ...A_n$ are attribute names, $T_1, T_2, ...T_n$ are attribute types).

Function Type: $F_{id}(P_1 : T_2, P_2 : T_2, ...P_n : T_n) \rightarrow T$ (F_{id} is a function name, $P_1, P_2, ...P_n$ are parameters, $T_1, T_2, ...T_n$ are parameter types T is the return type).

Safe Statement: $void \times t$ (t is a valid type capturing the return)

Unsafe Statement: $\alpha \times t$ (t is a valid type capturing the return)

Figure 60: Types of Types

Definition A type t is safe if:

t = int or bool

t = $*ref(id,t')$ or t = $ref(id,t')$ and t' is safe

$S_d(t) = S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$ and t_i is safe $\forall t \in t_1, t_2, ...t_n$

t = $void \times T$ and T is safe

Definition A type t is unsafe if:

t = α

t = $*ref(id,t')$ or t = $ref(id,t')$ and t' is unsafe

$S_d(t) = S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$ and $\exists t_i \in t_1, t_2, ...t_n$ such that t_i is unsafe

t = $\alpha \times T$

Definition: Subtyping: t1 <: t2, t1 is a subtype of t2
Definition: alloc $alloc(H,V,id,v) = H',V'$, where $H'(V'(id)) = v$
$alloc(H,v) = H',l$, where $H' = H[v/l]$ and $l \notin domain(H)$

Definition: realloc $realloc(H,v) = H',l$, where $H(l) = v'$ and $H' = H[v''/l]$. The value of v'' depends on v and v':

If $v = int\{i\}$ or $v = bool\{b\}$, then $v'' = v$.

If $v = S_{id}(...\{l_1, l_2, ...l_n\})$ and $v' = S_{id}(...\{l'_1, l'_2, ...l'_n\})$, then $v'' =$
$S_{id}(...\{l_1, l_2, ...l_n, l'_1, l'_2, ...l'_n\}$.

Definition: clear(H, V, id, l) $clear(H,V,id,l) = H',V'$ V' = V/id, if $l \notin domain(H)$, then $H' = H$. If $H'(l) = v$, H' depends on the value of v :

If $v = int\{i\}$ or $v = bool\{b\}$, then $H' = H/l$

If $v = S_{id}(...\{l_1, l_2, ...l_n\})$, then $H'' = dealloc(H, l_1, l_2, ...l_n)$ and $H' = H''/l$.

Definition: dealloc $dealloc(H,l) = H'$, if $l \notin domain(H)$, then $H' = H$. If $H'(l) = v$, H' depends on the value of v :

If $v = int\{i\}$ or $v = bool\{b\}$, then $H' = H/l$

If $v = S_{id}(...\{l_1, l_2, ...l_n\})$, then $H'' = dealloc(H, l_1, l_2, ...l_n)$ and $H' = H''/l$.

Figure 61: Definitions

Other Definitions

Define Substitution: $S[A \mapsto B] = S'$ where every instance of A is replaced with B in statement S.

Variables: $S[id \mapsto v] = S'$

Structures: $S[S_{id}\{A_1 = E_1,\ A_2 = E_2, ...,\ A_n = E_n\} \mapsto StructLiteral\{S_{id}\{A_1 = E_1,\ A_2 = E_2, ...,\ A_n = E_n\}\}]$

Functions: $S[F_{id1}(a_1, a_2, ...a_n) \mapsto FunctionLiteral\{(let\ P_1 : T_1 = a_1; let\ P_2 : T_2 = a_2\ ...\ let\ P_n : T_n = a_n; S_1 S_2 ... S_m\ return\ e;)\}]$

Definition Well-Foundedness: $R, E \vdash H, V$ if $\forall id \in domain(R)$ if:

$R(id) = t$ and $t \in \{int, bool\}$, and $H(V(id)) = v$ and $R, E \vdash v : t$

$R(id) = t = S_{id}(a_1 : t_1, a2 : t_2, ...a_n : t_n)$, and $H(V(id)) = v$, $R, E \vdash v : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$, $H(V(id)) = v = S_{id}(a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...an : t_n = l_n, ...)$, $R, E \vdash H(l_i) : t_i$, $\forall l_i \in \{l_1, l_2, ...l_n\}$ and $t_i \in \{t_1, t_2, ...t_n\}$

Definition Empty-Context: If $R, E \vdash e : t$, and R and E = $\emptyset$, then $\vdash e : t$

Figure 62: More Definitions

Abbreviations

$$alloc(H, v_1, v_2, ...v_n) = H', l_1, l_2, ...l_n, \text{ where:}$$
$$alloc(H, v_1) = H_1, l_1 \quad (H_1 = H[v_1/l_1])$$
$$alloc(H_1, v_2) = H_2, l_2 \quad (H_2 = H[v_2/l_2])$$
$$\vdots$$
$$alloc(H_{n-1}, v_n) = H', l_n \quad (H' = H[v_n/l_n])$$

$$dealloc(H, l_1, l_2, ...l_n) = H', \text{ where:}$$
$$dealloc(H, l_1) = H_1$$
$$dealloc(H_1, l_2) = H_2$$
$$\vdots$$
$$dealloc(H_{n-1}, l_n) = H'$$

$$clear(H, V, id_1, l_1, id_2, l_2, ...id_n, l_n) = H', V', \text{ where:}$$
$$clear(H, V, id_1, l_1) = H_1, V_1$$
$$clear(H_1, V_1, id_2, l_2) = H_2, V_2$$
$$\vdots$$
$$clear(H_{n-1}, V_{n-1}, id_n, l_n) = H', V'$$

Figure 63: Abbreviations B

6.2 Type Rules

Simple Expressions

$$\textbf{(TS-E-1)}\ \frac{i \text{ is a valid integer}}{R, E \vdash i : int} \qquad \frac{b \text{ is a valid boolean}}{R, E \vdash b : bool}\ \textbf{(TS-E-2)}$$

$$\textbf{(TS-E-3)}\ \frac{\begin{array}{c} R, E \vdash i_1 : int \qquad R, E \vdash i_2 : int \\ \text{op} \in \{+, -, *\} \end{array}}{R, E \vdash i_1\ op\ i_2 : int} \qquad \frac{}{t <: \alpha}\ \textbf{(TS-E-4)}$$

$$\frac{t \text{ is unsafe}}{R, E \vdash Error : ErrorType}\ \textbf{(TS-E-5)}$$

Figure 64: Simple Expression Typing Rules B

Variable Declarations

$$\textbf{(TS-V-1)}\ \frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin domain(R) \cup domain(E) \\ R, E \vdash e : \ t \\ t \text{ is safe} \qquad t \neq ref(id', t') \\ R' = R[t/id] \\ R', E \vdash S : t'' \times t'''\end{array}}{R, E \vdash let\ id : t\ =\ e; S : t'' \times t'''}\qquad \frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin domain(R) \cup domain(E) \\ R, E \vdash e : t \\ t \text{ is unsafe} \qquad t \neq ref(id', t') \\ E'_s = E[t/id] \\ R, E' \vdash S : t'' \times t'''\end{array}}{R, E \vdash let\ id : t\ =\ e; S\ : \alpha \times t'''}\ \textbf{(TS-V-2)}$$

$$\frac{\begin{array}{c}id \text{ is a valid identifier} \\ id \notin domain(R) \cup domain(E) \\ R, E \vdash e : t \\ t \neq ref(id', t') \\ E' = E[\alpha/id] \\ R, E \vdash S : t'' \times t'''\end{array}}{R, E \vdash let\ id : \alpha\ =\ e; S\ : \alpha \times t'''}\ \textbf{(TS-V-3)}$$

Figure 65: Variable Typing Rules

Assignment

$$\textbf{(TS-A-1)}\ \frac{R(id)\ =\ t \qquad R, E \vdash e : \ t \quad t \text{ is safe}}{R, E \vdash id\ =\ e\ ; : void \times void}\qquad \frac{R, E \vdash e : ref(id', t) \qquad id \neq id' \qquad R(id) = t \quad t \text{ is safe}}{R, E \vdash id\ =\ e\ ; : void \times void}\ \textbf{(TS-A-2)}$$

$$\textbf{(TS-A-3)}\ \frac{E(id) = t' \qquad R, E \vdash e : t}{R, E \vdash id\ =\ e\ ; : \alpha \times void}\qquad \frac{R, E \vdash ref(id', t) \qquad id \neq id' \qquad E(id) = t' \quad t \text{ is unsafe}}{F_d, S_d, R_s, E_s \vdash id\ =\ e\ ; : \alpha \times void}\ \textbf{(TS-V-A-4)}$$

Reference

$$\textbf{(TS-R-1)}\ \frac{\begin{array}{c}R(id) = t\ \vee\ E(id) = t \\ (\text{For the last case})\ H(l) = v \wedge R, E \vdash v : t\end{array}}{\begin{array}{c}R, E \vdash id\ : *ref(id, t) \\ R, E \vdash \&id\ : ref(id, t) \\ R, E \vdash ref\{id, l\}\ : ref(id, t)\end{array}}\qquad \frac{}{t <: *ref(id, t)}\ \textbf{(TS-R-2)}$$

$$\frac{R(id) = t\ \vee\ E(id) = t \qquad R, E \vdash v : t}{R, E \vdash *ref(id, v) : *ref(id, t)}\ \textbf{(TS-R-3)}$$

Figure 66: Other Variable Typing Rules B

Control Flow

$$\textbf{(TS-C-1)}\ \frac{\begin{array}{c} R,E \vdash b\ : bool \\ R,E \vdash S_1\ : void \times t \\ R,E \vdash S_2\ : void \times t \end{array}}{R,E \vdash if\ b\ \{S_1\}\ else\ \{S_2\} : void \times t}\qquad \frac{\begin{array}{c} R,E \vdash b\ : t_1 \\ R,E \vdash S_1\ : t_2 \times t' \\ R,E \vdash S_2\ : t_3 \times t'' \\ (t_1 = bool \vee t_1 = \alpha)\ \wedge t' \neq t'' \end{array}}{R,E \vdash if\ b\ \{S_1\}\ else\ \{S_2\} : \alpha \times \alpha}\ \textbf{(TS-C-1a)}$$

$$\textbf{(TS-C-2)}\ \frac{\begin{array}{c} R,E \vdash b\ : t_1 \\ R,E \vdash S_1\ : t_2 \times t' \\ R,E \vdash S_2\ : t_2 \times t' \\ t_1 = \alpha \vee (t_1 = bool \wedge t_2 = \alpha) \end{array}}{R,E \vdash if\ b\ \{S_1\}\ else\ \{S_2\} : \alpha \times t'}\qquad \textbf{(TS-C-3)}\ \frac{R,E \vdash b\ : bool R,E \vdash S\ : void \times t}{R,E \vdash while\ b\ \{S\} : void \times t}$$

$$\textbf{(TS-C-4)}\ \frac{\begin{array}{c} R,E \vdash b\ : t \\ R,E \vdash S\ : t' \times t'' \\ t = \alpha \vee (t = bool \wedge t' = \alpha) \end{array}}{R,E \vdash while\ b\ \{S\} : \alpha \times t''}\qquad \textbf{(TS-C-5)}\ \frac{\begin{array}{c} R,E \vdash S_1\ : t \times t' \\ R,E \vdash S_2\ : t \times t' \\ S_1 \text{ isn't a let statement} \end{array}}{R,E \vdash S_1\ S_2 : t \times t'}$$

$$\frac{\begin{array}{c} R,E \vdash S_1\ : t_1 \times t' \\ R,E \vdash S_2\ : t_2 \times t'' \\ t_1 = \alpha \vee t_2 = \alpha \vee t' \neq t'' \\ S_1 \text{ isn't a let statement} \end{array}}{R,E \vdash S_1\ S_2 : \alpha \times \alpha}\ \textbf{(TS-C-6)}$$

$$\textbf{(TS-C-7)}\ \frac{}{void \times void <: T_1 \times T_2}\qquad \frac{}{\alpha \times void <: \alpha \times T}\ \textbf{(TS-C-8)}\qquad \frac{}{ErrorType <: \alpha \times T}\ \textbf{(TS-C-8)}$$

Figure 67: Control Flow Typing Rules

Return

$$\textbf{(TS-R-1)}\ \frac{\begin{array}{c}R, E \vdash e : t \\ t \text{ is safe}\end{array}}{R, E \vdash return\ e;\ : void \times t}\quad \frac{\begin{array}{c}R, E \vdash e : t \\ t \text{ is unsafe}\end{array}}{R, E \vdash return\ e;\ : \alpha \times t}\ \textbf{(TS-R-2)}$$

Other

$$\textbf{(TS-O-1)}\ \frac{R, E \vdash e : t \qquad t \text{ is safe}}{R, E \vdash e; : void \times void}\quad \frac{R, E \vdash e : t \qquad t \text{ is unsafe}}{R, E \vdash e; : \alpha \times void}\ \textbf{(TS-O-1a)}$$

$$\textbf{(TS-O-2)}\ \frac{E(id) = t}{R, E \vdash free(id); : \alpha \times void}\quad \frac{\begin{array}{c}V(id) = l' \wedge H(l) = v \\ R, E \vdash v : t \qquad t \text{ is safe}\end{array}}{R, E \vdash clear(id, l); : void \times void}\ \textbf{(TS-O-3)}$$

$$\frac{R, E \vdash St[id \mapsto H(V(id))] : T_1 \times T_2}{R, E \vdash St : T_1 \times T_2}\ \textbf{(TS-O-4)}$$

Figure 68: Other Statements

Functions

$$\textbf{(TS-F-1)}\ \frac{\begin{array}{c}F_d(F_{id}) = F_{id}(P_1 : T_1, P_2 : T_2, ...P_n : T_n) \to T_r \\ P_1,\ P_2,\ ...,\ P_n \text{ are unique} \\ \forall t \in T_1,\ T_2,\ ...,\ T_n,\ T_r : t \text{ is safe} \\ R' = \{P_1 : T_1, P_2 : T_2, ..., P_n : T_n\} \\ R', E \vdash S_1\ S_2\ ...\ S_j\ return\ e; \\ clear(id_1, l_1); clear(id_2, l_2); ...clear(id_n, l_n); : void \times T' \\ T' = T_r \wedge T' \neq ref(id, T'') \\ j, n \geq 0\end{array}}{\begin{array}{c}R, E \vdash def\ F_{id}(P_1 : T_1,\ P_2 : T_2, ...\ P_n : T_n) \to \\ T_r\ \{S_1\ S_2\ ...\ S_j\ return\ e; clear(id_1, l_1); clear(id_2, l_2); ...clear(id_n, l_n);\} \\ : F_{id}(P_1 : T_1,\ P_2 : T_2, ...\ P_n : T_n) \to T_r\end{array}}$$

$$\textbf{(TS-F-1a)}\ \frac{\begin{array}{c}F_d(F_{id}) = F_{id}(P_1 : T_1, P_2 : T_2, ...P_n : T_n) \to T_r \\ P_1,\ P_2,\ ...,\ P_n \text{ are unique} \\ \forall t \in T_{a1},\ T_{a2},\ ...,\ T_{ab} : t \text{ is safe} \\ \forall t \in T_{c1},\ T_{c2},\ ...,\ T_{cd}, T_r : t \text{ is unsafe} \\ b, d, n \geq 0 \qquad b + d = n \\ T_{a1},\ T_{a2},\ ...,\ T_{ab} \cup T_{c1},\ T_{c2},\ ...,\ T_{cd} = T_1,\ T_2,\ ...,\ T_n \\ R' = \{P_{a1} : T_{a1}, P_{a2} : T_{a2}, ..., P_{ab} : T_{ab}\} \\ E' = \{P_{c1} : T_{c1}, P_{c2} : T_{c2}, ..., P_{cd} : T_{cd}\} \\ R', E' \vdash S_1\ S_2\ ...\ S_j\ return\ e; clear(id_1, l_1); clear(id_2, l_2); ...clear(id_n, l_n);: \\ \alpha \times T_r' T_r = \alpha \vee T_r' = T_r\end{array}}{\begin{array}{c}R, E \vdash def\ F_{id}(P_1 : T_1,\ P_2 : T_2, ...\ P_n : T_n) \to \\ T_r\ \{S_1\ S_2\ ...\ S_j\ return\ e; clear(id_1, l_1); clear(id_2, l_2); \\ ...clear(id_n, l_n);\} : F_{id}(P_1 : T_1,\ P_2 : T_2, ...\ P_n : T_n) \to T_r\end{array}}$$

$$\textbf{(TS-F-2)}\ \frac{R, E \vdash St : T_1 \times T_2 \wedge T_1 \text{ isn't } \alpha}{R, E \vdash FunctionLiteral\{St\} : T_2}$$

$$\textbf{(TS-F-3)}\ \frac{R, E \vdash St : T_1 \times T_2 \wedge T_1 \text{ isn't } void}{R, E \vdash FunctionLiteral\{St\} : \alpha}$$

Figure 69: Function Typing Rules

Structures

$$\textbf{(TS-S-1)}\ \frac{R,E \vdash St : e_1 : t_1,\quad R,E \vdash St : e_2 : t_2,\ ...,\ R,E \vdash St : e_n : t_n}{R,E \vdash StructureLiteral\{S_{id}\{a_1 = e_1, a_2 = e_2, ...a_n = e_n\}\} : S_{id}\{a_1 : t_1, a_2 : t_2, ...a_n : t_n\}}$$

$$\textbf{(TS-S-2)}\ \frac{R,E \vdash St : e_1 : t_1,\quad R,E \vdash St : e_2 : t_2,\ ...,\ R,E \vdash St : e_n : t_n}{R,E \vdash S_{id}\{a_1 = e_1, a_2 = e_2, ...a_n = e_n\} : S_{id}\{a_1 : t_1, a_2 : t_2, ...a_n : t_n\}}$$

$$\textbf{(TS-S-3)}\ \frac{l_1, l_2, ...l_n \in domain(H)}{R,E \vdash S_{id}\{a_1 : l_1, a_2 : l_2, ...a_n : l_n\} : S_{id}\{a_1 : t_1, a_2 : t_2, ...a_n : t_n\}}$$

Structure Assignment

$$\textbf{(TS-S-A-1)}\ \frac{\begin{array}{c} R,E \vdash e_1 : *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}) \\ \forall t \in t_1, t_2, ...\ t_n : t \text{ is safe} \qquad id = A_i \in A_1, A_2, ...A_n \\ R,E \vdash e_2 : T_i \qquad t_i \neq ref(id'', t) \end{array}}{R,E \vdash e_1.id = e_2 : void \times void}$$

$$\textbf{(TS-S-A-2)}\ \frac{\begin{array}{c} R,E \vdash e_1 : *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}) \\ \forall t \in t_1, t_2, ...\ t_n : t \text{ is safe} \qquad id = a_i \in a_1, a_2, ...a_n \\ R,E \vdash e_2 : ref(id'', t_i) \\ R(id') = S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\} \qquad R(id'') = t_i \end{array}}{F_d, S_d, R_s, E_s \vdash e_1.id = e_2 : void \times void}$$

$$\textbf{(TS-S-A-3)}\ \frac{\begin{array}{c} R,E \vdash e_1 : *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}) \\ id = a_i \in a_1, a_2, ...a_n \qquad t_i \text{ is unsafe} \\ R,E \vdash e_2 : t_i' \qquad t_i' = t_i \vee t_i = \alpha \qquad t_i' \neq ref(id'', t) \end{array}}{R,E \vdash e_1.id = e_2 : \alpha \times void}$$

$$\textbf{(TS-S-A-4)}\ \frac{\begin{array}{c} R,E \vdash e_1 : *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}) \\ id = a_i \in a_1, a_2, ...a_n \qquad t_i \text{ is unsafe} \\ R,E \vdash e_2 : ref(id', t_i') \qquad t_i = \alpha \vee t_i' = t_i \end{array}}{R,E \vdash e_1.id = e_2 : \alpha \times void}$$

$$\textbf{(TS-S-A-5)}\ \frac{\begin{array}{c} R,E \vdash e_1 : *ref(id', \alpha) \\ R,E \vdash e_2 : t \\ t \neq ref(id'', t') \end{array}}{R,E \vdash e_1.id = e_2 : \alpha \times void} \qquad \frac{\begin{array}{c} R,E \vdash e_1 : *ref(id', \alpha) \\ R,E \vdash e_2 : ref(id'', t) \\ t \text{ is unsafe} \end{array}}{R,E \vdash e_1.id = e_2 : \alpha \times void}\ \textbf{(TS-S-A-6)}$$

Figure 70: Structure Typing Rules

More Structures

$$\textbf{(TS-S-R-1)}\ \frac{\begin{array}{c} R, E \vdash e_1 : *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}) \\ id = a_i \in a_1, a_2, ...a_n \end{array}}{\begin{array}{c} R, E \vdash e_1.id : *ref(id', t_i) \\ R, E \vdash \&e_1.id : ref(id', t_i) \end{array}}$$

$$\textbf{(TS-S-R-1)}\ \frac{R, E \vdash e_1 : *ref(id', \alpha)}{R, E \vdash e_1.id : \alpha}$$

Figure 71: More Structure Typing Rules

Global Definitions

$$\textbf{(TS-G-1)}\ \frac{\begin{array}{c}
R, E \vdash def\ F_{id1}(P_{11} : T_{11},\ P_{12} : T_{12}, ...,\ P_{1n_1} : T_{1n_1}) \to T_{r1}\ \{S_{11}S_{12}...S_{1x_1}\ return\ e_1;\} : T'_1 \\
R, E \vdash def\ F_{id2}(P_{21} : T_{21},\ P_{22} : T_{22}, ...,\ P_{2n_2} : T_{2n_2}) \to T_{r2}\ \{S_{21}S_{22}...S_{2x_2}\ \ return\ e_2;\} : T'_2 \\
\vdots \\
R, E \vdash def\ F_{idj}(P_{j1} : T_{j1},\ P_{j2} : T_{j2}, ...,\ P_{jn_j} : T_{jn_j}) \to T_{rj}\ \{S_{j1}S_{j2}...S_{jx_j}\ \ return\ e_j;\} : T'_j \\
R, E \vdash struct\ S_{id1}\{A_{11} : T''_{11},\ A_{12} : T''_{12}, ...,\ A_{1m_1} : T''_{1m_1}\} : T'_{j+1} \\
R, E \vdash struct\ S_{id2}\{A_{21} : T''_{21},\ A_{22} : T''_{22}, ...,\ A_{2m_2} : T''_{2m_2}\} : T'_{j+2} \\
\vdots \\
R, E \vdash struct\ S_{idk}\{A_{k1} : T''_{k1},\ A_{k2} : T''_{k2}, ...,\ A_{km_k} : T''_{km_k}\}\ S : T'_{j+k} \\
R, E \vdash\ S[F_{id1}(a_{11}, a_{12}, ...a_{1n_1}) \mapsto FunctionLiteral \\
\{(let\ P_{11} : T_{11} = a_{11}; let\ P_{12} : T_{12} = a_{12}\ ...\ let\ P_{1n} : T_{1n} = a_{1n}; S_{11}S_{22}...S_{x_1}\ return\ \ \ e_1;)\}] \\
[F_{id2}(a_{21}, a_{22}, ...a_{2n_2}) \mapsto FunctionLiteral \\
\{(let\ P_{21} : T_{21} = a_{21}; let\ P_{22} : T_{22} = a_{22}\ ...\ let\ P_{2n} : T_{2n} = a_{2n}; S_{21}S_{22}...S_{2x_2}\ return\ \ \ e_2;)\}]... \\
[F_{idn}(a_{j1}, a_{j2}, ...a_{jn_j}) \mapsto FunctionLiteral \\
\{(let\ P_{j1} : T_{j1} = a_{j1}; let\ P_{j2} : T_{j2} = a_{j2}\ ...\ let\ P_{jn_j} : T_{jn_j} = a_{n_j}; S_{j1}S_{j2}...S_{x_j}\ return\ \ \ e_j;)\}] \\
[S_{id1}\{A_{11} = E_{11},\ A_{12} = E_{12}, ...,\ A_{1m_1} = E_{1m_1}\} \mapsto \\
StructLiteral\{S_{id1}\{A_{11} = E_{11},\ A_{12} = E_{12}, ...,\ A_{1m_1} = E_{1m_1}\}\}] \\
[S_{id2}\{A_{21} = E_{21},\ A_{22} = E_{22}, ...,\ A_{2m_2} = E_{2m_2}\} \mapsto \\
StructLiteral\{S_{id2}\{A_{21} = E_{21},\ A_{22} = E_{22}, ...,\ A_{2m_2} = E_{2m_2}\}\}]... \\
[S_{idk}\{A_{k1} = E_{k1},\ A_{k2} = E_{k2}, ...,\ A_{km_k} = E_{km_k}\} \mapsto \\
StructLiteral\{S_{idk}\{A_{k1} = E_{k1},\ A_{k2} = E_{k2}, ...,\ A_{km_k} = E_{km_k}\}\}] : t \\
0 \leq j, k \\
0 \leq n_1, n_2, ...n_j \\
0 \leq x_1, x_2, x_3, ..., x_j \\
0 \leq m_1, m_2, ...m_k \\
S_{id} \text{ are unique} \qquad F_{id} \text{ are unique} \\
A_i,\ P_i \text{ are unique in the context of their definitions}
\end{array}}{\begin{array}{c}
R, E \vdash def\ F_{id1}(P_{11} : T_{11},\ P_{12} : T_{12}, ...,\ P_{1n_1} : T_{1n_1}) \to T_{r1}\ \{S_{11}S_{12}...S_{1x_1}\ return\ e_1;\} \\
def\ F_{id2}(P_{21} : T_{21},\ P_{22} : T_{22}, ...,\ P_{2n_2} : T_{2n_2}) \to T_{r2}\ \{S_{21}S_{22}...S_{2x_1}\ return\ e_2;\} \\
\vdots \\
def\ F_{idj}(P_{j1} : T_{j1},\ P_{j2} : T_{j2}, ...,\ P_{jn_j}\ T_{jn_j}) \to T_{rj}\ \{S_{j1}S_{j2}...S_{jx_j}\ return\ e_j;\} \\
struct\ S_{id1}\{A_{11} : T''_{11},\ A_{12} : T''_{12}, ...,\ A_{1m_1} : T''_{1m_1}\} \\
struct\ S_{id2}\{A_{21} : T''_{21},\ A_{22} : T''_{22}, ...,\ A_{2m_2} : T''_{2m_2}\} \\
\vdots \\
struct\ S_{idk}\{A_{k1} : T''_{k1},\ A_{k2} : T''_{k2}, ...,\ A_{km_k} : T''_{km_k}\}\ S : t
\end{array}}$$

Figure 72: Global Definitions Typing Rules B

6.3 Small-step Operational Semantics

Simple Expressions

$$\textbf{(S-E-1)}\ \frac{e_1, H \longrightarrow e', H', V' \quad \text{op} \in \{+,\text{-},*\}}{e_1\ op\ e_2, H, V \longrightarrow\ e'\ op\ e_2, H', V'} \qquad \frac{e, H \longrightarrow e', H', V' \quad \text{op} \in \{+,\text{-},*\} \quad n \in integers}{n\ op\ e, H, V \longrightarrow\ n\ op\ e', H', V'}\ \textbf{(S-E-1b)}$$

$$\textbf{(S-E-1c)}\ \frac{\text{op} \in \{+,\text{-},*\} \quad n_3 = n_1\ op\ n_2 \quad n_1, n_2, n_3 \in integers}{n_1\ op\ n_2, H, V \longrightarrow\ n_3, H, V}$$

$$\textbf{(S-E-1d)}\ \frac{\text{op} \in \{+,\text{-},*\}}{\text{Error}\ op\ e, H, V \longrightarrow\ \text{Error}, H, V} \qquad \frac{\text{op} \in \{+,\text{-},*\}}{n\ op\ \text{Error}, H, V \longrightarrow\ \text{Error}, H, V}\ \textbf{(S-E-1e)}$$

$$\textbf{(S-E-1f)}\ \frac{\text{op} \in \{+,\text{-},*\} \quad n \in literals \wedge n \notin integers}{n\ op\ e, H, V \longrightarrow\ \text{Error}, H, V}$$

$$\textbf{(S-E-1g)}\ \frac{\text{op} \in \{+,\text{-},*\} \quad m \in literals \wedge m \notin integers}{n\ op\ m, H, V \longrightarrow\ \text{Error}, H, V}$$

Figure 73: Simple Expression Evaluation Rules

Variable Declarations

$$\textbf{(S-V-D-1)}\ \frac{e, H, V \longrightarrow e', H', V'}{let\ id : t\ =\ e; St, H, V \longrightarrow let\ id : t\ =\ e'; St, H', V'}$$

$$\textbf{(S-V-D-2)}\ \frac{t \text{ is safe} \qquad H', V', l = alloc(H, V, id, v)}{let\ id : t\ =\ v; St, H, V \longrightarrow St\ \ clear(id, l);, H', V'}$$

$$\textbf{(S-V-D-3)}\ \frac{}{let\ id : t\ = Error; StH, V \longrightarrow Error, H, V}$$

$$\frac{t \text{ is unsafe} \qquad H', V' = alloc(H, V, id, v)}{let\ id : t\ =\ v; St, H, V \longrightarrow St, H', V'}\ \textbf{(S-V-D-4)}$$

Figure 74: Variable Declaration Evaluation Rules B

Assignments

$$\textbf{(S-V-A-1)}\ \frac{e, H', V' \longrightarrow e', H', V'}{id\ =\ e\ ;, H, V \longrightarrow id = e';, H', V} \qquad \frac{H' = realloc(H, V(id))]}{id\ =\ v\ ;, H \longrightarrow \_, H', V}\ \textbf{(S-V-A-2)}$$

$$\textbf{(S-V-A-3)}\ \frac{V' = V[id/l]}{id\ =\ ref\{id', l\}\ , H, V' \longrightarrow \_H, V'} \qquad \frac{}{id = Error, H, V; \longrightarrow Error, H, V}\ \textbf{(S-V-A-3)}$$

References

$$\textbf{(S-V-R-1)}\ \frac{V(id) = l \qquad l \in domain(H) \qquad H(l) = v}{id, H, V \longrightarrow *ref(id, v)H, V} \qquad \frac{}{*ref(id, v), H, V \longrightarrow v, H, V}\ \textbf{(S-V-R-2)}$$

$$\textbf{(S-V-R-3)}\ \frac{V(id) = l \qquad l \notin domain(H) \qquad H(l) = v}{id, H, V \longrightarrow Error, H, V} \qquad \frac{V(id) = l \qquad l \in domain(H)}{\&id, H, V \longrightarrow ref\{id, l\}, H, V}\ \textbf{(S-V-R-4)}$$

$$\frac{V(id) = l \qquad l \notin domain(H)}{\&id, H, V \longrightarrow Error, H, V}\ \textbf{(S-V-R-5)}$$

Figure 75: Other Variable Evaluation Rules

Statements

$$\textbf{(S-S-1)}\ \frac{S_1, H, V \longrightarrow S', H', V' \qquad (S_1 \neq let\ id : t = v; S_1 \neq return\ e;)}{S_1 S_2 \longrightarrow S' S_2, H', V'}$$

$$\textbf{(S-S-2)}\ \frac{}{\_S_2, H, V \longrightarrow S_2, H, V} \qquad \frac{}{return\ v; S, H, V \longrightarrow return\ v;, H, V}\ \textbf{(S-S-3)}$$

$$\frac{}{Error; S, H, V \longrightarrow Error;, H, V}\ \textbf{(S-S-4)}$$

Control Flow

$$\textbf{(S-C-1)}\ \frac{b, H, V \longrightarrow b', H', V'}{if\ b\ \{S_1\}\ \{S_2\}, H, V \longrightarrow if\ b'\ \{S_1\}\ \{S_2\}, H', V'}$$

$$\textbf{(S-C-2)}\ \frac{}{if\ true\ \{S_1\}\ \{S_2\}, H, V \longrightarrow S_1, H, V}$$

$$\frac{}{if\ false\ \{S_1\}\ \{S_2\}, H, V \longrightarrow S_2, H, V}\ \textbf{(S-C-3)}$$

$$\frac{}{if\ Error\ \{S_1\}\ \{S_2\}, H, V \longrightarrow Error, H, V}\ \textbf{(S-C-4)}$$

$$\textbf{(S-C-5)}\ \frac{}{while\ b\ \{St\}, H, V \longrightarrow if\ b\ \{St\ while\ b\ \{St\}\}\ else\{St\}, H, V}$$

Return

$$\textbf{(S-R-1)}\ \frac{e, H, V \longrightarrow e', H', V'}{return\ e;, H, V \longrightarrow return\ e';, H', V'}$$

$$\frac{}{return\ Error;, H, V \longrightarrow Error, H, V}\ \textbf{(S-R-2)}$$

Figure 76: Statement Evaluation Rules

Other Statements

$$\textbf{(S-O-1)}\ \frac{e,H,V \longrightarrow e',H',V'}{e;,H,V \longrightarrow e';,H',V'} \qquad \frac{}{v;,H,V \longrightarrow \_,H,V}\ \textbf{(S-O-2)}$$

$$\textbf{(S-O-3)}\ \frac{}{Error;,H,V \longrightarrow Error,H,V} \qquad \frac{H' = H/V(id) \qquad V' = V/id}{free(id),H,V \longrightarrow \_,H',V'}\ \textbf{(S-O-4)}$$

$$\textbf{(S-O-5)}\ \frac{id \notin domain(V) \vee V(id) \notin domain(H)}{free(id),H,V \longrightarrow Error,H,V} \qquad \frac{H',V' = clear(H,V,id_1,l_2)}{clear(id_1,l_1);,H,V \longrightarrow \_,H',V'}\ \textbf{(S-O-6)}$$

Figure 77: Other Statement Evaluation Rules

Functions

$$\textbf{(S-F-1)}\ \frac{St,H,V \longrightarrow St',H',V'}{FunctionLiteral\{St\},H,V \longrightarrow FunctionLiteral\{St'\},H',V'}$$

$$\frac{H',V' = clear(H,V,id_1,l_1,id_2,l_2,...id_n,l_n);}{\begin{array}{c}FunctionLiteral\{return\ v;\ clear(id_1,l_1);\\ clear(id_2,l_2);...clear(id_n;l_n);\},H,V \longrightarrow v,H',V'\end{array}}\ \textbf{(S-F-2)}$$

$$\frac{n \geq 0}{\begin{array}{c}FunctionLiteral\{Error\ clear(id_1,l_1);\\ clear(id_2,l_2);...clear(id_n,l_n);\},H,V \longrightarrow Error,H,V\end{array}}\ \textbf{(S-F-3)}$$

Figure 78: Function Evaluation Rules

Structures

$$\textbf{(S-S-L-1)}\ \frac{}{StructLiteral\{S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}, H, V}$$

$$\textbf{(S-S-L-2)}\ \frac{e_{i+1}, H, V \longrightarrow e'_{i+1}, H', V'}{\begin{array}{c} S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i... \\ a_{i+1} : t_{i+1} = e_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H, V \\ \longrightarrow \\ S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = \\ v_i...a_{i+1} : t_{i+1} = e'_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H', V' \end{array}}$$

$$\textbf{(S-S-L-3)}\ \frac{alloc(H, v_1, v_2, ...v_n) = H', l_1, l_2, ...l_n}{S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_n : t_n = v_n\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}, H', V}$$

$$\textbf{(S-S-L-4)}\ \frac{}{\begin{array}{c} S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i... \\ a_{i+1} : t_{i+1} = Error, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, \\ H, V \longrightarrow Error, H, V \end{array}}$$

Figure 79: Structure Evaluation Rules

Structure Assignments

$$\textbf{(S-S-A-1)}\ \frac{e_1, H, V \longrightarrow e_1', H', V'}{e_1.id = e_2, H, V \longrightarrow e_1'.id = e_2, H', V'}$$

$$\frac{e, H, V \longrightarrow e', H', V'}{\begin{array}{c} *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e, H, V \longrightarrow \\ *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e', H', V' \end{array}}\ \textbf{(S-S-A-2)}$$

$$\textbf{(S-S-A-3)}\ \frac{id' = a_i \qquad H' = realloc(H, v)}{*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = v, H, V \longrightarrow \_, H', V}$$

$$\textbf{(S-S-A-4)}\ \frac{\begin{array}{c} id' = a_i \\ V' = V[S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i\ ...\ a_n : t_n = l_n\}/id]) \\ id \in domain(V) \end{array}}{\begin{array}{c} *ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i\ ...\ a_n : t_n = l_n\}).id' \\ = ref\{id'', l\}, H, V \longrightarrow \_, H, V' \end{array}}$$

$$\textbf{(S-S-A-5)}\ \frac{\begin{array}{c} (v_1 = *ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i\ ...\ a_n : t_n = l_n\}) \wedge \\ id \notin domain(V)) \vee \\ (v_1 \neq *ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i\ ...\ a_n : t_n = l_n\})) \vee \\ (v_2 = ref\{id, l\} \wedge l \notin domain(H)) \vee (v_2 = Error) \end{array}}{v_1.id' = v_2, H, V \longrightarrow Error, H, V}$$

Figure 80: Structure Assignment Evaluation Rules B

Structure References

$$\textbf{(S-S-R-1)}\ \frac{e, H, V \longrightarrow e', H', V'}{e.id, H, V \longrightarrow e'.id, H', V'}$$

$$\frac{id' = a_i \qquad l_i \in domain(H) \qquad H(l_i) = v}{\begin{array}{c} *ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ... \\ a_i : t_i = l_i \ ... \ a_n : t_n = l_n\}).id', H, V \longrightarrow v, H, V \end{array}}\ \textbf{(S-S-R-2)}$$

$$\frac{\begin{array}{c} id' = a_i \qquad H(l) = S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i \ ... \ a_n : t_n = l_n\} \\ l_i \in domain(H) \end{array}}{ref\{id, l\}.id', H, V \longrightarrow ref\{id, l_i\}, H, V}\ \textbf{(S-S-R-4)}$$

$$\textbf{(S-S-R-5)}\ \frac{\begin{array}{c} (v = *ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i \ ... \ a_n : t_n = l_n\}) \wedge \\ id' = a_i, l_i \notin domain(H)) \vee \\ (v \neq *ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i \ ... \ a_n : t_n = l_n\}) \wedge \\ (v \neq ref\{id, l\})) \vee (v = ref\{id, l\} \wedge \\ (H(l) \neq S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i \ ... \ a_n : t_n = l_n\} \vee \\ id = a_i, l_i \notin domain(H))) \end{array}}{v.id', H, V \longrightarrow Error, H, V}$$

Figure 81: Structure Reference Evaluation Rules B

Global Definitions

(S-G-1)

$$
\begin{array}{c}
def\ F_{id1}(P_{11}:T_{11},\ P_{12}:T_{12},...,\ P_{1n_1}:T_{1n_1}) \to T_{r1}\ \{S_{11}S_{12}...S_{1x_1}\ return\ e_1;\} \\
def\ F_{id2}(P_{21}:T_{21},\ P_{22}:T_{22},...,\ P_{2n_2}:T_{2n_2}) \to T_{r2}\ \{S_{21}S_{22}...S_{2x_1}\ return\ e_2;\} \\
\vdots \\
def\ F_{idj}(P_{j1}:T_{j1},\ P_{j2}:T_{j2},...,\ P_{jn_j}T_{jn_j}) \to T_{rj}\ \{S_{j1}S_{j2}...S_{jx_j}\ return\ e_j;\} \\
struct\ S_{id1}\{A_{11}:T''_{11},\ A_{12}:T''_{12},...,\ A_{1m_1}:T''_{1m_1}\} \\
struct\ S_{id2}\{A_{21}:T''_{21},\ A_{22}:T''_{22},...,\ A_{2m_2}:T''_{2m_2}\} \\
\vdots \\
struct\ S_{idk}\{A_{k1}:T''_{k1},\ A_{k2}:T''_{k2},...,\ A_{km_k}:T''_{km_k}\}, St \longrightarrow \\
St[F_{id1}(a_{11},a_{12},...a_{1n_1}) \mapsto FunctionLiteral \\
\{(let\ P_{11}:T_{11}=a_{11};let\ P_{12}:T_{12}=a_{12}\ ...\ let\ P_{1n}:T_{1n}=a_{1n};S_{11}S_{22}...S_{x_1}\ return \quad e_1;)\}] \\
[F_{id2}(a_{21},a_{22},...a_{2n_2}) \mapsto FunctionLiteral \\
\{(let\ P_{21}:T_{21}=a_{21};let\ P_{22}:T_{22}=a_{22}\ ...\ let\ P_{2n}:T_{2n}=a_{2n};S_{21}S_{22}...S_{2x_2}\ return \quad e_2;)\}]... \\
[F_{idn}(a_{j1},a_{j2},...a_{jn_j}) \mapsto FunctionLiteral \\
\{(let\ P_{j1}:T_{j1}=a_{j1};let\ P_{j2}:T_{j2}=a_{j2}\ ...\ let\ P_{jn_j}:T_{jn_j}=a_{n_j};S_{j1}S_{j2}...S_{x_j}\ return \quad e_j;)\}] \\
[S_{id1}\{A_{11}=E_{11},\ A_{12}=E_{12},...,\ A_{1m_1}=E_{1m_1}\} \mapsto \\
StructLiteral\{S_{id1}\{A_{11}=E_{11},\ A_{12}=E_{12},...,\ A_{1m_1}=E_{1m_1}\}\}] \\
[S_{id2}\{A_{21}=E_{21},\ A_{22}=E_{22},...,\ A_{2m_2}=E_{2m_2}\} \mapsto \\
StructLiteral\{S_{id2}\{A_{21}=E_{21},\ A_{22}=E_{22},...,\ A_{2m_2}=E_{2m_2}\}\}]... \\
[S_{idk}\{A_{k1}=E_{k1},\ A_{k2}=E_{k2},...,\ A_{km_k}=E_{km_k}\} \mapsto \\
StructLiteral\{S_{idk}\{A_{k1}=E_{k1},\ A_{k2}=E_{k2},...,\ A_{km_k}=E_{km_k}\}\}]
\end{array}
$$

Figure 82: Global Definitions Evaluation Rules B

6.4 Clear Assumption

Clear is a special statement. If this language had a compiler, it would insert clear statements in an intermediate stage. Clear isn't meant to be a part of the surface language, so I assume some special rules for its use. Firstly, I assume that all clear statements are valid only if they result from the evaluation rule **(S-O-6)**. From this, I assume that clears only exist at the natural end of a variable lifetime, meaning for all instances of $clear(id,l);$, $V(id) = l'$ and $H(l) = v$. Lastly, I also assume that clears only appear in expressions where $id \notin domain(R)$. These are simple and fair assumptions based on the semantic and typing rules, but annoying to incorporate because clear is an intermediate form.

7 Soundness and Safety

7.1 Progress

Theorem 7.1 Progress 1

Progress 1 Part 1: If $\vdash$ E : T and T is safe, then E is an integer, a boolean, a safe reference literal, a safe structure literal, or there exists some E', such that $\mathrm{E}, H, V \longrightarrow E', H', V'$.

Proof by strong induction on the depth of the derivation of $\vdash E : T$, *(and* $\vdash E : void \times T_2$, *see Progress Part 2). Consider the last step of the derivation:*

Case 1: $\vdash$ *i:* i is an integer.

Case 2: $\vdash$ *b:* b is a boolean.

Case 3: $\vdash e_1$ *op* $e_2 : int$ *and op* $\in \{+, -, *\}$

1. e_1 *op* e_2: By inductive assumption, $e_1, H, V \longrightarrow e_1', H', V'$. So, e_1 *op* $e_2, H, V \longrightarrow e_1', H', V'$ *op* e_2 **(S-E-1)**
2. n *op* e: By inductive assumption, $e, H, V \longrightarrow e', H', V'$. So, n *op* $e, H, V \longrightarrow$ n *op* e', H', V' **(S-E-2)**
3. *n op m,H,V* $\longrightarrow$ *v,H,V* ***(S-E-1c)***

Case 4: $\vdash id : t_1, \vdash *ref(id, v) : t_2, \vdash \&id : t_3$ and $\vdash ref\{id, l\} : t_4$

1. *id:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.
2. **ref(id, v)* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.
3. *&id:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.
4. *ref{id, l}* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

Figure 83: Progress 1 Part 1 A

Continuation 1 Progress 1 Continued

Case 5: $\vdash FunctionLiteral\{St\} : T$

1. $FunctionLiteral\{St\}, H, V \longrightarrow FunctionLiteral\{St'\}, H', V'$: By inductive assumption, $St, H, V \longrightarrow St', H', V'$. So, $FunctionLiteral\{St\}, H, V \longrightarrow FunctionLiteral\{St'\}, H', V'$. ***(S-F-1)***

2. $FunctionLiteral\{return \quad v; clear(id_1, l_1); clear(id_2, l_2); ...clear(id_n, l_n);\}, H, V \longrightarrow v, H, V'$ ***(S-F-2)***

Case 6: $\vdash S_{id}\{a_1 : t_1 = E_1, a_2 : t_2 = E_2, ...a_n : t_n = E_n\}\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$ and structure literal.

1. $StructrueLiteral\{S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}, H, V$.***(S-S-L-1)***

2. By inductive assumption, $e_i, H, V \longrightarrow e'_{i+1}, H', V'$. So, $StructrueLiteral\{S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i_n} = e_{i+n}\}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e'_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H, V$ ***(S-S-L-2)***

3. $S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i\}, H', V$.***(S-S-L-3)***

4. $S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}$ is a structure literal.

Case 7: $\vdash e.id$

1. $e.id$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

2. $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id'$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

3. $ref\{id, l\}.id'$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

Figure 84: Progress 1 Part 1 B

Continuation 2 Progress 1 Continued

Progress 1 Part 2: If $\vdash$ E : $void \times T$, then E is _, or there exists some E', such that $E, H, V \longrightarrow E', H', V'$.

Case 1: $\vdash let\ id : t = e; St : T_1 \times T_2$

1. *let id:t = e; St:* By inductive assumption, $e, H, V \longrightarrow e', H', V'$. So, *let id:t = e;St*, $H, V \longrightarrow let\ id : t = e'; St, H', V'$ ***(S-V-D-1)***.

2. *let id:t = v; St, H, V* $\longrightarrow$ *St clear*$(id, l);, H', V'$ ***(S-V-D-2)***
   $(H', V', l = alloc(H, V, id, v))$

Case 2: $\vdash id = e; : T_1 \times T_2$

1. *id = e;:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

2. *id = v;:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

3. *id* = ref{id', l};: This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

Case 3: $\vdash S_1 S_2 : T_1 \times T_2$

1. $S_1 S_2$ ($S_1 \neq let\ id : t = v; S_1 \neq return\ e;$)*:* By inductive assumption, $S_1, H, V \longrightarrow S_1', H', V'$. *So,* $S_1 S_2, H, V \longrightarrow S_1' S_2, H', V'$ ***(S-S-1)***

2. _ $S_2, H, V \longrightarrow S_2, H, V$ ***(S-S-2)***

3. *return* $v; S_2, H, V \longrightarrow return\ v;, H, V$ ***(S-S-3)***

Case 4: $\vdash return\ e; : T_1 \times T_2$

1. *return e;:* By inductive assumption $e, H, V \longrightarrow e', H', V'$. So, *return e;*, $H, V \longrightarrow return\ e';, H', V'$ ***(S-R-1)***

Case 5: $\vdash if\ b\ \{S_1\}\ else\ \{S_2\} : T_1 \times T_2$

1. *if* $b\ \{S_1\}$ *else* $\{S_2\}$*:* By inductive assumption $b, H, V \longrightarrow b', H', V'$. *So, if* $b\ \{S_1\}$ *else* $\{S_2\}, H, V \longrightarrow if\ b'\ \{S_1\}$ *else* $\{S_2\}, H', V'$ ***(S-C-1)***

2. *if true* $\{S_1\}$ *else* $\{S_2\}, H, V \longrightarrow S_1, H, V$ ***(S-C-2)***

3. *if false* $\{S_1\}$ *else* $\{S_2\}, H, V \longrightarrow S_2, H, V$ ***(S-C-3)***

Case 6: $\vdash while\ b\ \{S_1\} : T_1 \times T_2$

1. *while* $b\ \{St\}, H, V \longrightarrow if\ b\ \{St\ while\ b\ \{St\}\}\ else\ \{St\}, H, V$ ***(S-C-5)***

Figure 85: Progress 1 Part 2 A

Continuation 3 Progress 1 Continued

Case 7: $\vdash e_1.id = e_2$

1. $e_1.id = e_2$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

2. $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = e$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

3. $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = v$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

4. $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = ref\{id'', l\}$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

Case 8: $\vdash$ ***Global Definitions***

1. **Global Definitions** reduce. ***(S-G-1)*** I don't list it here, because it is clear they reduce by rule S-G-1 and it is a very large rule.

Case 9: $\vdash e;$

1. $e;, H \longrightarrow e';, H', V'$: By inductive assumption, $e, H \longrightarrow e', H', V'$, so $e;, H \longrightarrow e';, H', V'$:***(S-O-1)***

2. $v;, H, V \longrightarrow \_, H, V$ ***(S-O-2)***

Case 10: $\vdash free(id);$

1. $free(id);$ This is impossible. Since E_s is empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

Case 11: $\vdash clear(id, l);$

1. $clear(id, l);$ This is impossible. Since R_s and E_s are empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

Figure 86: Progress 1 Part 2 B

Theorem 7.2 Progress 2

Progress 2 Part 1: If $\vdash$ E : T and T is safe, then E is an integer, a boolean, a reference literal, a structure literal, or there exists some E', such that $\mathrm{E}, H, V \longrightarrow E', H', V'$.

Proof by strong induction on the depth of the derivation of $\vdash E : T$, *(and* $\Gamma \vdash E : void \times T_2$, *see Progress 2 Part 2). Consider the last step of the derivation:*

Case 1: $\vdash i$: i is an integer.

Case 2: $\vdash b$: b is a boolean.

Case 3: $\vdash e_1\ op\ e_2 : int$ *and* $op \in \{+, -, *\}$

1. $e_1\ op\ e_2$: By inductive assumption, $e_1, H, V \longrightarrow e_1', H', V'$. So, $e_1\ op\ e_2, H, V \longrightarrow e_1', H', V'\ op\ e_2$ **(S-E-1)**
2. n *op* e: By inductive assumption, $e, H, V \longrightarrow e', H', V'$. So, n *op* $e, H, V \longrightarrow$ n *op* e', H', V' **(S-E-2)**
3. $n\ op\ m, H, V \longrightarrow v, H, V$ ***(S-E-1c)***
4. $Error\ op\ e_2, H, V \longrightarrow Error, H, V$ ***(S-E-1d)***
5. $n\ op\ Error, H, V \longrightarrow Error, H, V$ ***(S-E-1e)***
6. $v\ op\ e_2, H, V \longrightarrow Error, H, V$ ***(S-E-1f)***
7. $n\ op\ v, H, V \longrightarrow Error, H, V$ ***(S-E-1g)***

Case 4: $\vdash id : t, \vdash *ref(id, v) : t_2, \vdash \&id : t_3$ *and* $\vdash ref\{id, l\} : t_4$

1. *id:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.
2. **ref(id, v):* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.
3. *&id:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.
4. *ref{id, l}:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

Figure 87: Progress 2 Part 1 A

Continuation 4

Case 5: $\vdash FunctionLiteral\{St\} : T$

1. $FunctionLiteral\{St\}, H, V \longrightarrow FunctionLiteral\{St\}, H', V'$: By inductive assumption, $St, H, V \longrightarrow St', H', V'$. So, $FunctionLiteral\{St\}, H, V \longrightarrow FunctionLiteral\{St'\}, H', V'$. ***(S-F-1)***

2. $FunctionLiteral\{return\ v;\ clear(id_1, l_1); clear(id_2, l_2); ...clear(id_n, l_n);\}, H, V \longrightarrow v, H', V'$ ***(S-F-2)***

3. $FunctionLiteral\{Error\ clear(id_1, id_2, ...id_n);\}, H, V \longrightarrow Error, H, V$ ***(S-F-3)***

Case 6: $\vdash S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\} : S_{id}(a_1 = e_1, t_2 = t_2, ...a_n = t_n)$

1. $StructrueLiteral\{S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}, H, V$. ***(S-S-L-1)***

2. By inductive assumption, $e_i, H, V \longrightarrow e'_{i+1}, H', V'$. So, $StructrueLiteral\{S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e'_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H, V$ ***(S-S-L-2)***

3. $S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_i : t_i = l_i\}, H', V$. ***(S-S-L-3)***

4. $S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = Error, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H, V \longrightarrow Error, H, V$ ***(S-S-L-4)***

Case 7: $\vdash e.id$

1. $e.id$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

2. $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id'$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

3. $ref\{id, l\}.id'$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

Figure 88: Progress 1 Part 1 B

Continuation 5

Progress 2 Part 2: If $\vdash$ E : $\alpha \times T$, then E is _, Error, or there exists some E', such that $\mathrm{E}, H, V \longrightarrow E', H', V'$.

Case 1: $\vdash let\ id : t = e; St : T_1 \times T_2$

1. *let id:t = e; St:* By inductive assumption, $e, H, V \longrightarrow e', H', V'$. So, *let id:t = e;St*, $H, V \longrightarrow let\ id : t = e'; St, H', V'$ ***(S-V-D-1)***.

2. *let id:t = v; St, H, V* $\longrightarrow St\ clear(id, l);, H', V'$ ***(S-V-D-2)***
   $(H', V', l = alloc(H, V, id, v))$

3. *let id:t = Error; St: let id:t = Error;St*, $H, V \longrightarrow Error, H, V$ ***(S-V-D-3)***.

4. *let id:t = v; St, H, V* $\longrightarrow St;, H', V'$ ***(S-V-D-4)***
   $(H', V' = alloc(H, V, id, v))$

Case 2: $\vdash id = e; : T_1 \times T_2$

1. *id = e;:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

2. *id = v;:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

3. *id =* ref{id', l};: This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

4. *id = Error;:* This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

Case 3: $\vdash S_1 S_2 : T_1 \times T_2$

1. $S_1 S_2$ $(S_1 \neq let\ id : t = v; S_1 \neq return\ e;)$*:* By inductive assumption, $S_1, H, V \longrightarrow S_1', H', V'$. *So,* $S_1 S_2, H, V \longrightarrow S_1' S_2, H', V'$ ***(S-S-1)***

2. _ $S_2, H, V \longrightarrow S_2, H, V$ ***(S-S-2)***

3. $return\ v; S_2, H, V \longrightarrow return\ v;, H, V$ ***(S-S-3)***

4. $Error\ St, H, V \longrightarrow Error, H, V$ ***(S-S-4)***

Case 4: $\vdash return\ e; : T_1 \times T_2$

1. *return e;:* By inductive assumption $e, H, V \longrightarrow e', H', V'$. So, $return\ e;, H, V \longrightarrow return\ e';, H', V'$ ***(S-R-1)***

2. *return Error;*, $H, V \longrightarrow Error, H, V$ ***(S-R-2)***

Figure 89: Progress 2 Part 2 A

Continuation 6

Case 5: $\vdash if\ b\ \{S_1\}\ else\ \{S_2\} : T_1 \times T_2$

1. *if* $b\ \{S_1\}\ else\ \{S_2\}$*:* By inductive assumption $b, H, V \longrightarrow b', H', V'$. *So, if* $b\ \{S_1\}\ else\ \{S_2\}, H, V \longrightarrow if\ b'\ \{S_1\}\ else\ \{S_2\}, H', V'$ ***(S-C-1)***
2. *if true* $\{S_1\}\ else\ \{S_2\}, H, V \longrightarrow S_1, H, V$ ***(S-C-2)***
3. *if false* $\{S_1\}\ else\ \{S_2\}, H, V \longrightarrow S_2, H, V$ ***(S-C-3)***
4. *if Error* $\{S_1\}\ else\ \{S_2\}, H, V \longrightarrow Error, H, V$ ***(S-C-4)***

Case 6: $\vdash while\ b\ \{S_1\} : T_1 \times T_2$

1. $while\ b\ \{St\}, H, V \longrightarrow if\ b\ \{St\ while\ b\ \{St\}\}\ else\ \{St\}, H, V$ ***(S-C-5)***

Case 7: $\vdash e_1.id = e_2$

1. $e_1.id = e_2$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$
2. $*ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$
3. $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = v$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$
4. $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = ref\{id'', l\}$ This is impossible. Since R_s *and* E_s are both empty, this cannot possibly be the last derivation; $id \notin \emptyset$

Figure 90: Progress 2 Part 2 B

Continuation 7

Case 8: $\vdash$ ***Global Definitions***

1. **Global Definitions** reduce. ***(S-G-1)*** I don't list them here, because it is clear they reduce by rule S-G-1 and it is a very large rule.

Case 9: $\vdash e;$

1. $e;, H \longrightarrow e';, H', V'$: By inductive assumption, $e, H \longrightarrow e', H', V'$, so $e;, H \longrightarrow e';, H', V'$:***(S-O-1)***
2. $v;, H, V \longrightarrow \_, H, V$ ***(S-O-2)***
3. $Error;, H, V \longrightarrow Error, H, V$ ***(S-O-3)***

Case 10: $\vdash free(id);$

1. $free(id)$;This is impossible. Since E_s is empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

Case 11: $\vdash clear(id, l);$

1. $clear(id, l)$; This is impossible. Since R_s and E_s are empty, this cannot possibly be the last derivation; $id \notin \emptyset$.

Figure 91: Progress 2 Part 2 C

8 Preservation

8.1 Preliminaries

Statement Reduction Lemma: For all St, where $\Gamma \vdash St : T_1 \times T_2$, St = _ or $St, H, V \longrightarrow St', H', V'$.

Proof: I give a reduction for every possible statement while proving preservation. For the sake of brevity, I omit them here; just note all statements are either _ or they reduce. This Lemma for statements with nested statements.

Underscore Lemma: $\Gamma \vdash \_ : T_1 \times T_2$

Proof: Since $\Gamma \vdash \_ : void \times void$, by **(TC-C-7)** $\Gamma \vdash \_ : T_1 \times T_2$.

8.2 Preservation Theorems

Theorem 8.1 Preservation 1

Preservation 1 Part 1: Let Γ be some typing context: R, E. If $\Gamma \vdash E : T$ and T is safe, $\Gamma \vdash H, V$ and $E, H, V \longrightarrow E', H', V'$. Then $\Gamma \vdash E' : T$ and $\Gamma \vdash H', V'$.

Proof by strong induction on the depth of the derivation of $\Gamma \vdash E : T$, *(and* $\Gamma \vdash E : void \times T_2$, *see Preservation 1 Part 2). Consider the last step of the derivation:*

Case 1: i is an integer, thus it cannot be reduced.

Case 2: b is a boolean, thus it cannot be reduced.

Figure 92: Preservation 1 Part 1 A

Continuation 8

Case 3: $e_1\ op\ e_2\ op \in \{+, -, *\}$

$\Gamma \vdash E_1\ op\ E_2 :\ int$, and by inversion of the typing rules, $\Gamma \vdash E_1$: int and $\Gamma \vdash E_2$: int.

Consider the possible reductions:

1. ***(S-E-1)*** $e_1\ op\ e_2, H, V \longrightarrow e_1'\ op\ e_2, H', V'$

   $e_1, H, V \longrightarrow e_1', H', V'$ (by inversion of the evaluation rule). So, by inductive assumption, $\Gamma \vdash e_1'$ *: int* and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash e_1'\ op\ e_2$ *: int.*

2. ***(S-E-1b)*** $n\ op\ e, H, V \longrightarrow n\ op\ e, H', V'$

   $e, H, V \longrightarrow e', H', V'$ (by inversion of the evaluation rule). So, by inductive assumption, $\Gamma \vdash e'$ *: int* and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash n\ op\ e_2'$ *: int.*

3. ***(S-E-1c)*** $n\ op\ m\ , H, V \longrightarrow v,\ H, V$. I assume the basic laws of arithmetic, thus n op m is an integer. $\Gamma \vdash v : int$ ***(TS-E-1)***

Case 4: $id,\ *ref(id, v),\ \&id,$and $ref\{id, l\}$

Since $\Gamma \vdash id : T_1, \Gamma \vdash *ref(id, v) : T_2, \Gamma \vdash \&id : T_3$ and $\Gamma \vdash ref\{id, l\} : T_4$, and $\Gamma \vdash H, V$, also $H(V(id)) = v$

Consider the possible reductions:

1. $\Gamma \vdash id : *ref(id, t),$ then $id, H, V \longrightarrow *ref(id, v), H, V$ **(S-V-R-1)**. $\Gamma \vdash *ref(id, v) : *ref(id, t)$ **(TS-R-3)**

2. $\Gamma \vdash *ref(id, v) : t$ **(TS-R-2)**, so $*ref(id, v), H, V \longrightarrow v, H, V$ **(S-V-R-2)** and $\Gamma \vdash$ v:t

3. $\Gamma \vdash \&id : ref(id, t),$ then $\&id, H, V \longrightarrow ref\{id, v\}, H, V$ **(S-V-R-4)**. $\Gamma \vdash ref\{id, v\} : ref(id, t)$ **(TS-R-1)**

4. $ref\{id, v\}$, doesn't reduce.

Figure 93: Preservation 1 Part 1 B

Continuation 9

Case 5: $\Gamma \vdash FunctionLiteral\{St\} : T$, by inversion of the type rules, $\Gamma \vdash \text{St} : void \times T_2$

1. $FunctionLiteral\{St\}, H, V \longrightarrow FunctionLiteral\{St'\}, H', V'$ ***(S-F-1)***

   $St, H, V \longrightarrow St', H', V'$ ***(Statement Reduction Lemma)***. By inductive assumption (the second part of the assumption, see Preservation 1 Part 2), $\Gamma \vdash St' : void \times T_2$ and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash FunctionLiteral\{St'\} : T$.

2. $FunctionLiteral\{return\ v; clear(id, l); clear(id_2, l_2); ...clear(id_n; clear_n);\}, H, V \longrightarrow v, H', V'$ ***(S-F-2)***.

   Recall, $\Gamma \vdash return\ v; : void \times T_2$, *and* T_2 must be the same type as v, by inversion of the typing judgement for return **(TS-R-1)**;. Thus, $\Gamma \vdash v : T$. Also, $\Gamma \vdash H', V'$. By the typing premises **(TS-F-2)** H(l) = v, $\forall l \in l_1, l_2, ...l_n$, and $V(id) = l$ and $R(id) \neq t$, $\forall id \in id_1, id_2, ...id_n$. So, $\Gamma \vdash H', V'$ where $H', V' = clear(H, V, id_1, l_1, id_2, l_2, ...l_n, id_n)$

Case 6: $\Gamma \vdash StructLiteral\{S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$, by inversion of the typing rules, $\Gamma \vdash S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$

1. $StructLiteral\{S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : tn = e_n\}, H, V$ **(S-S-L-1)**. Recall, $\Gamma \vdash S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$

Case 7: $\Gamma \vdash S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_i : t_i ...a_{i+1} : t_{i+1}, a_{i+2} : t_{i+2}, ...a_{i+n} : t_{i+n})$. Also, by inversion of the typing rules, $\Gamma \vdash e_{i+1} : t_{i+1}$.

1. $e_{i+1}, H, V \longrightarrow e'_{i+1}, H', V'$, so by inductive assumption $\Gamma \vdash e'_{i+1} : t_{i+1}$ and $\Gamma \vdash H', V'$. $S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e'_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H', V'$.**(S-S-L-2)** Thus, $\Gamma \vdash S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e'_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_i = t_i ...a_{i+1} : t_{i+1}, a_{i+2} : t_{i+2}, ...a_{i+n} : t_{i+n})$

Case 8: $\Gamma \vdash S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_n : t_n = v_n\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$

1. ***(S-S-L-3)*** $S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_n : t_n = v_n\} \longrightarrow S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}, H', V$. From the rule, $H' = alloc(H, v_1, v_2, ...v_n)$ Since, $\Gamma \vdash H, l1, l2, ...ln \notin domain(H)$ and $H' = H[v_1/l_1][v_2/l_2]...[v_n/l_n]\Gamma \vdash H', V$.

Case 9: $\Gamma \vdash S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$

1. $S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}$ doesn't reduce.

Figure 94: Preservation 1 Part 1 B

Continuation 10

Case 10: $\Gamma \vdash e.id : t, \Gamma \vdash e : T$

1. ***(S-S-R-1)*** $e.id, H, V \longrightarrow e'.id, H', V'$

   $e, H, V \longrightarrow e', H', V'$, by inductive assumption, $\Gamma \vdash e' : T$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash e'.id : t$.

2. ***(S-S-R-2)*** $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id', H, V \longrightarrow v, H, V$. Since $\Gamma \vdash$ H, H(l_i) = v and $\Gamma \vdash v : t$

3. ***(S-S-R-3)*** $ref\{id, l\}.id', H, V \longrightarrow ref\{id, l_i\}, H, V$. Note $\Gamma \vdash ref\{id, l\}.id' : ref(id, t)$. Since $\Gamma \vdash$ H, H(l_i) = v and $\Gamma \vdash v : t$. Thus, $\Gamma \vdash ref\{id, l_i\} : ref(id, t)$.**(TS-R-1)**

Figure 95: Preservation 1 Part 1 C

Continuation 11

Preservation 1 Part 2: Let Γ be some typing context: F_d, S_d, R_s, E_s. If $\Gamma \vdash E : void \times T$, $\Gamma \vdash H, V$ and $E, H, V \longrightarrow E', H', V'$. Then $\Gamma \vdash E' : void \times T$ and $\Gamma \vdash H', V'$.

Case 1: *let id:t = e; St*

$\Gamma \vdash$ let id:t = e; St *:* $void \times T$, by inversion of the rules $\Gamma \vdash e : T_e$ *and* $\Gamma \vdash St : void \times T$

Consider the possible reductions:

1. ***(S-V-D-1)*** $let\ id{:}t = e;\ St, H, V \longrightarrow let\ id{:}t = e'; St, H', V'$

   $e, H, V \longrightarrow e', H', V'$. By Preservation Part 1 $\Gamma \vdash e' : T_e$ and $\Gamma \vdash H', V'$. St didn't change, so $\Gamma \vdash St : void \times T$. Thus, $\Gamma \vdash let\ id : t = e'; St : void \times T$.

2. ***(S-V-D-2)*** $let\ id{:}t = v;\ St, H, V \longrightarrow St\ clear(id, l);, H', V'$ $(H', V', l = alloc(H, V, id, v))$. $\Gamma \vdash St : void \times T$ **(TS-O-4)** and $\Gamma \vdash clear(id, l);: void \times void$ **(TS-O-3)**. So, by **(TS-C-5)** and **(TS-C-7)**, $\Gamma \vdash St\ clear(id, l);: void \times T$. Also, $H' = H[v/l]$ and $V' = V[l/id]$, $l \notin domain(H)$ and $id \notin domain(V)$, so if $\Gamma \vdash H, V$, then $\Gamma \vdash H', V'$.

Figure 96: Preservation 1 Part 2 A

Continuation 12

Case 2: *id = e;*

$\Gamma \vdash$ id = e: $T_1 \times T_2$, by inversion of the rule, $\Gamma \vdash$ e: T_e

Consider the possible reductions:

1. ***(S-V-A-1)*** $id = e;, H, V \longrightarrow id = e';, H', V'$

   $e, H, V \longrightarrow e', H', V'$. By inductive assumption, $\Gamma \vdash e' : T_e$, and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash id = e';: T_1 \times T_2$.

2. ***(S-V-A-2)*** id = v;, $H, V \longrightarrow \_, H', V, \Gamma \vdash \_ : T_1 \times T_2$**(Underscore Lemma)**. Since $R(id) = t$ and $\Gamma \vdash H, V$ $H(V(id)) = v'$ and $\Gamma \vdash v' : t$. Since $H' = H[v/l], H(V(id)) = v$ and $\Gamma \vdash v : t$, so $\Gamma \vdash H', V$.

3. ***(S-V-A-3)*** id = ref{id', l};, $H, V \longrightarrow \_, H, V', \Gamma \vdash \_ : T_1 \times T_2$**(Underscore Lemma)**. Since $R(id) = t$ and $\Gamma \vdash H, V$, $V(id) = l'$, $H(l') = v$ and $\Gamma \vdash v : t$. By inversion of the typing rules $H(l) = v$ and $\Gamma \vdash v : t$, so $\Gamma \vdash H, V'$.

Case 3: $S_1 S_2$

$\Gamma \vdash S_1 S_2$: $T_1 \times T_2$, by inversion of the rules, $\Gamma \vdash S_1$: $T_3 \times T_4$, and $\Gamma \vdash S_2$: $T_5 \times T_6$

Consider the possible reductions:

1. ***(S-S-1)*** $S_1 S_2, H, V \longrightarrow, H', V'$ *($S_1 \neq let\ id : t = v; S_1 \neq return\ e;$)*

   $S_1, H, V \longrightarrow S_1', H', V'$ ***(Statement Reduction Lemma)***. So, by inductive assumption, $\Gamma \vdash S_1' : T_3 \times T_4$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash S_1' S_2 : T_1 \times T_2$.

2. ***(S-S-2)*** $\_\ S_2, H, V \longrightarrow S_2, H, V$, $\Gamma \vdash\ S_2 : T_1 \times T_2$ ***(TS-C-5)***

3. ***(S-S-3)*** $return\ v; S_2, H, V \longrightarrow return\ v;, H, V : \Gamma \vdash return\ v;: T_1 \times T_2$. (Either return is the same type as the original expression, or ***(TS-C-7)*** is applicable).

Case 4: *return e;* $\Gamma \vdash return\ e;: T_1 \times T_2$, by inversion of the rules $\Gamma \vdash e : T_e$

1. ***(S-R-1)*** $return\ e;, H, V \longrightarrow return\ e';, H', V'$

   $e, H, V \longrightarrow e', H', V'$. So, by inductive assumption $\Gamma \vdash e : T_e$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash return\ e';: T_1 \times T_2$.

Figure 97: Preservation 1 Part 2 B

Continuation 13

Case 5: $\Gamma \vdash if\ b\ \{S_1\}\ else\ \{S_2\} : T_1 \times T_2$. By inversion of the typing rule $\Gamma \vdash b : T_e$, $\Gamma \vdash S_1 : T_3 \times T_4$, and $\Gamma \vdash S_2 : T_5 \times T_6$

1. ***(S-C-1)*** $if\ b\ \{S_1\}\ else\ \{S_2\}, H, V \longrightarrow if\ b'\ \{S_1\}\ else\ \{S_2\}, H', V'$

   $b, H, V \longrightarrow b', H', V'$, so by inductive assumption $\Gamma \vdash b' : T_e$ and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash if\ b'\ \{S_1\}\ else\ \{S_2\} : T_1 \times T_2$.

2. ***(S-C-2)*** $if\ true\ \{S_1\}\ else\ \{S_2\}, H, V \longrightarrow S_1, H, V$. (Either S_1 is the same type as the original expression, or **(TS-C-7)** is applicable).

3. ***(S-C-2)*** $if\ false\ \{S_1\}\ else\ \{S_2\}, H, V \longrightarrow S_1, H, V$. (Either S_1 is the same type as the original expression, or **(TS-C-7)** is applicable).

Case 6: $\Gamma \vdash while\ b\ \{S_1\} : T_1 \times T_2$

1. ***(S-C-6)*** $while\ b\ \{St\}, H, V \longrightarrow if\ b\ \{St\ while\ b\ \{St\}\}\ else\{S_t\},\ H, V$

   Since they share the same condition b and the same statement St,
   $\Gamma \vdash if\ b\ \{St\ while\ b\ \{St\}\}\ else\{St\} : T_1 \times T_2$.

Case 7: $\Gamma \vdash e_1.id = e_2 : T \times void$, $\Gamma \vdash e_1 : t_1, \Gamma \vdash e_2 : t_2$

1. ***(S-S-A-1)*** $e_1.id = e_2, H, V \longrightarrow e_1'.id = e_2, H', V'$

   $e_1, H, V \longrightarrow e_1', H', V'$, by inductive assumption, $\Gamma \vdash e_1' : t_1$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash e_1'.id = e_2 : T \times void$.

2. ***(S-S-A-2)*** $*ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e, H, V \longrightarrow *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e', H', V'$

   $e_2, H, V \longrightarrow e_2', H', V'$, by inductive assumption, $\Gamma \vdash e_2' : t_2$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e' : T \times void$.

3. ***(S-S-A-3)***$*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = v, H, V \longrightarrow \_, H', V'$

   $\Gamma \vdash \_ : T \times void$**(Underscore Lemma)**. Since $\Gamma \vdash H$, $H(l_i) = v'$ and $\Gamma \vdash v' : t_i$. By inversion of the typing rules $\Gamma \vdash v : t_i$, also $H' = H[v/l]$, so $\Gamma \vdash H', V$

4. ***(S-S-A-4)*** $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = ref\{id'', l\}, H, V \longrightarrow \_, H', V'$

   $\Gamma \vdash \_ : T \times void$ **(Underscore Lemma)**. Since $\Gamma \vdash H, H(l_i) = v'$ and $\Gamma \vdash v' : t_i$. By inversion of the typing rules, $H(l) = v$ and $\Gamma \vdash v : t_i$. $V' = V[id/l]$, so $\Gamma \vdash H', V'$.

Figure 98: Preservation 1 Part 2 C

Continuation 14

Case 8: $\Gamma \vdash$ **Global Definitions** St

1. ***(S-G-1)*** **Global Definitions** $St \longrightarrow St'$ ***(Substitution Assumption)***

Case 9: $\Gamma \vdash e; : void \times void$. By inversion of the rules, $\Gamma \vdash e : t$

1. ***(S-O-1)*** $e;, H \longrightarrow e';, H', V'$: By inductive assumption, $\Gamma \vdash e : t$, and $\Gamma \vdash H', V'$, so $\Gamma \vdash e; : void \times void$.

2. $v;, H, V \longrightarrow \_, H, V$ $\Gamma \vdash \_ : void \times void$.***(Underscore Lemma)***

Case 11: $\Gamma \vdash clear(id, l); : void \times void$

1. $clear(id, void);, H, V \longrightarrow \_, H', V'$: $\Gamma \vdash \_ : void \times void$***(Underscore Lemma)*** *and* $\Gamma \vdash H', V'$ ***(Clear Assumption)***

Figure 99: Preservation 1 Part 2 D

Theorem 8.2 Preservation 2

Preservation 2 Part 1: Let Γ be some typing context: F_d, S_d, R_s, E_s. If $\Gamma \vdash E : T$ and T is unsafe, $\Gamma \vdash H, V$and $E, H, V \longrightarrow E', H', V'$. Then $\Gamma \vdash E' : T$ and $\Gamma \vdash H', V'$.

Proof by strong induction on the depth of the derivation of $\Gamma \vdash E : T$, *(and* $\Gamma \vdash E : \alpha \times T_2$, *see Preservation Part 2). Consider the last step of the derivation:*

Case 1: i is an integer, thus it cannot be reduced.

Case 2: b is a boolean, thus it cannot be reduced.

Figure 100: Preservation 2 Part 1 A

Continuation 15

Case 3: $e_1\ op\ e_2\ op \in \{+, -, *\}$,

$\Gamma \vdash E_1\ op\ E_2 : \alpha$, and by inversion of the typing rules, $\Gamma \vdash E_1$: T_1 and $\Gamma \vdash E_2$: T_2.

Consider the possible reductions:

1. ***(S-E-1)*** $e_1\ op\ e_2, H, V \longrightarrow e_1'\ op\ e_2, H', V'$

   $e_1, H, V \longrightarrow e_1', H', V'$ (by inversion of the evaluation rule). So, by inductive assumption, $\Gamma \vdash e_1' : T_1$ and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash e_1'\ op\ e_2 : \alpha$.

2. ***(S-E-1b)*** $n\ op\ e, H, V \longrightarrow n\ op\ e, H', V'$

   $e, H, V \longrightarrow e', H', V'$ (by inversion of the evaluation rule). So, by inductive assumption, $\Gamma \vdash e' : T_2$ and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash n\ op\ e_2' : \alpha$.

3. ***(S-E-1c)*** $n\ op\ m\ , H, V \longrightarrow v,\ H, V$. I assume the basic laws of arithmetic, thus n op m is an integer. So, $\Gamma \vdash v : int$, but by **(TS-E-4)**, $\Gamma \vdash v : \alpha$. (The type: int and everything else is a sub-type of alpha).

4. ***(S-E-1d)*** $Error\ op\ e_2, H, V \longrightarrow Error, H, V$. $\Gamma \vdash Error : \alpha$ ***(TS-E-5)***

5. ***(S-E-1e)*** $n\ op\ Error, H, V \longrightarrow Error, H, V$. $\Gamma \vdash Error : \alpha$ ***(TS-E-5)***

6. ***(S-E-1f)*** $v\ op\ e_2, H, V \longrightarrow Error, H, V$. $\Gamma \vdash Error : \alpha$ ***(TS-E-5)***

7. ***(S-E-1g)*** $n\ op\ v, H, V \longrightarrow Error, H, V$. $\Gamma \vdash Error : \alpha$ ***(TS-E-5)***

Figure 101: Preservation 2 Part 1 B

Continuation 16

Case 4: $id,\ *ref(id,v),\ \&id,$and $ref\{id,l\}$

Since $\Gamma \vdash id : T_1, \Gamma \vdash *ref(id,v) : T_2, \Gamma \vdash \&id : T_3$ and $\Gamma \vdash ref\{id,l\} : T_4$, and $\Gamma \vdash H, V$, then $H(V(id)) = v$

Consider the possible reductions:

1. $\Gamma \vdash id : *ref(id,t),$ then $id, H, V \longrightarrow *ref(id,v), H, V$**(S-V-R-1)**. $\Gamma \vdash *ref(id,v) : *ref(id,t)$**(TS-R-3)**
2. $\Gamma \vdash *ref(id,v) : t$ **(TS-R-2)**, so $*ref(id,v), H, V \longrightarrow v, H, V$ **(S-V-R-2)** and $\Gamma \vdash$ v:t
3. $\Gamma \vdash \&id : ref(id,t),$ then $\&id, H, V \longrightarrow ref\{id,v\}, H, V$**(S-V-R-4)**. $\Gamma \vdash ref\{id,v\} : ref(id,t)$**(TS-R-1)**
4. $ref\{id,v\}$, doesn't reduce.
5. $\Gamma \vdash id : t$***(TS-R-2)*** $id, H, V \longrightarrow Error, H, V$ ***(S-V-R-3)*** $\Gamma \vdash Error : t$***(TS-E-5)***
6. $\Gamma \vdash \&id : t$***(TS-R-2)*** $\&id, H, V \longrightarrow Error, H, V$ ***(S-V-R-5)*** $\Gamma \vdash Error : t$***(TS-E-5)***

Figure 102: Preservation 2 Part 1 C

Continuation 17

Case 5: $\Gamma \vdash FunctionLiteral\{St\} : T$, by inversion of the type rules, $\Gamma \vdash \text{St} : \alpha \times T_2$

1. $FunctionLiteral\{St\}, H, V \longrightarrow FunctionLiteral\{St'\}, H', V'$ ***(S-F-1)***

   $St, H, V \longrightarrow St', H', V'$ ***(Statement Reduction Lemma)***. By inductive assumption (the second part of the assumption, see Preservation Part 2), $\Gamma \vdash St' : \alpha \times T_2$ and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash FunctionLiteral\{St'\} : T$.

2. $FunctionLiteral\{return \quad v; clear(id_1, l_1); clear(id_2, l_2); ...clear(id_n, l_n);\}, H, V \longrightarrow v, H, V$ ***(S-F-2)*** . Recall, $\Gamma \vdash return\ v; : \alpha \times T_2$, *and* T_2 must be the same type as v, by inversion of the typing judgement for return **(TS-R-1)**, or . Thus, $\Gamma \vdash v : T$. *Also,* $\Gamma \vdash H', V'$ ***(Clear Assumption)***

Case 6: $\Gamma \vdash StructLiteral\{S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$, by inversion of the typing rules, $\Gamma \vdash S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$

1. $StructLiteral\{S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : tn = e_n\}, H, V$ **(S-S-L-1)**. Recall, $\Gamma \vdash S_{id}\{a_1 : t_1 = e_1, a_2 : t_2 = e_2, ...a_n : t_n = e_n\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$

Case 7: $\Gamma \vdash S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_i : t_i ...a_{i+1} : t_{i+1}, a_{i+2} : t_{i+2}, ...a_{i+n} : t_{i+n})$. Also, by inversion of the typing rules, $\Gamma \vdash e_{i+1} : t_{i+1}$.

1. $e_{i+1}, H, V \longrightarrow e'_{i+1}, H', V'$, so by inductive assumption $\Gamma \vdash e'_{i+1} : t_{i+1}$ and $\Gamma \vdash H', V'$. $S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H, V \longrightarrow S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e'_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H', V'$.**(S-S-L-2)** Thus, $\Gamma \vdash S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = e'_{i+1}, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_i = t_i ...a_{i+1} : t_{i+1}, a_{i+2} : t_{i+2}, ...a_{i+n} : t_{i+n})$

2. $S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_i : t_i = v_i ...a_{i+1} : t_{i+1} = Error, a_{i+2} : t_{i+2} = e_{i+2}, ...a_{i+n} : t_{i+n} = e_{i+n}\}, H, V \longrightarrow Error, H, V$.**(S-S-L-4)** Thus, $\Gamma \vdash Error : S_{id}(a_1 : t_1, a_2 : t_2, ...a_i = t_i ...a_{i+1} : t_{i+1}, a_{i+2} : t_{i+2}, ...a_{i+n} : t_{i+n})$ **(TS-E-4)**

Case 8: $\Gamma \vdash S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_n : t_n = v_n\} : S_{id}(a_1 : t_1, a_2 : t_2, ...a_n : t_n)$

1. ***(S-S-L-3)*** $S_{id}\{a_1 : t_1 = v_1, a_2 : t_2 = v_2, ...a_n : t_n = v_n\} \longrightarrow S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}, H', V$. From the rule, $H' = alloc(H, v_1, v_2, ...v_n)$ Also, $\Gamma \vdash H', V$ Since, $\Gamma \vdash H, l1, l2, ...ln \notin domain(H)$ and $H' = H[v_1/l_1][v_2/l_2]...[v_n/l_n]\Gamma \vdash H', V$.

Figure 103: Preservation 2 Part 1 D

Continuation 18

Case 9: $\Gamma \vdash e.id : t, \Gamma \vdash e : T$

1. ***(S-S-R-1)*** $e.id, H, V \longrightarrow e'.id, H', V'$

   $e, H, V \longrightarrow e', H', V'$, by inductive assumption, $\Gamma \vdash e' : T$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash e'.id : t$.

2. ***(S-S-R-2)*** $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id', H, V \longrightarrow v, H, V$. Since $\Gamma \vdash$ H, H(l_i) = v and $\Gamma \vdash v : t$

3. ***(S-S-R-3)*** $ref\{id, l\}.id', H, V \longrightarrow ref\{id, l_i\}, H, V$. Note $\Gamma \vdash ref\{id, l\}.id' : ref(id, t)$. Since $\Gamma \vdash$ H, H(l_i) = v and $\Gamma \vdash v : t$. Thus, $\Gamma \vdash ref\{id, l_i\} : ref(id, t)$.**TS-R-1**

4. **(S-S-R-5)** v.id', H, V $\longrightarrow Error, H, V$ Since, $\Gamma \vdash e.id : t$ and t is unsafe, $\Gamma \vdash Error : t$.**(TS-E-4)**

Figure 104: Preservation 2 Part 1 E

Theorem 8.3

Preservation 2 Part 2: Let Γ be some typing context: F_d, S_d, R_s, E_s. If $\Gamma \vdash E : \alpha \times T$, $\Gamma \vdash H, V$ and $E, H, V \longrightarrow E', H', V'$. Then $\Gamma \vdash E' : \alpha \times T$ and $\Gamma \vdash H', V'$.

Case 1: *let id:t = e; St*

$\Gamma \vdash$ let id:t = e; St *:* $\alpha \times T$, by inversion of the rules $\Gamma \vdash e : T_e$ *and* $\Gamma \vdash St : \alpha \times T$

Consider the possible reductions:

1. ***(S-V-D-1)*** *let id:t = e; St*, $H, V \longrightarrow$ *let id:t = e'; St*, H', V'

   $e, H, V \longrightarrow e', H', V'$. By Preservation 2 Part 1 $\Gamma \vdash e' : T_e$ and $\Gamma \vdash H', V'$. St didn't change, so $\Gamma \vdash St : \alpha \times T$. Thus, $\Gamma \vdash let\ id : t = e'; St : \alpha \times T$.

2. ***(S-V-D-2)*** *let id:t = v; St*, $H, V \longrightarrow St, H', V'$ $(H', V' = alloc(H, V, id, v))$. $\Gamma \vdash St : \alpha \times T$ **(TS-O-4)**. So, $\Gamma \vdash St : \alpha \times T$. Also, $H' = H[v/l]$ and $V' = V[l/id]$, $l \notin domain(H)$ and $id \notin domain(V)$, so if $\Gamma \vdash H, V$, then $\Gamma \vdash H', V'$.

3. ***(S-V-D-3)*** *let id:t = Error; St, H, V* $\longrightarrow$ *Error, H, V.* $\Gamma \vdash Error : \alpha \times T$. ***(TS-C-8)***

Figure 105: Preservation Part 2 A

Continuation 19

Case 2: *id = e;*

$\Gamma \vdash$ id = e: $T_1 \times T_2$, by inversion of the rule, $\Gamma \vdash$ e: T_e

Consider the possible reductions:

1. ***(S-V-A-1)*** *id = e;*, $H, V \longrightarrow id = e';, H', V'$

   e,$H, V \longrightarrow e', H', V'$. By inductive assumption, $\Gamma \vdash e' : T_e$, and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash id = e';: T_1 \times T_2$.

2. ***(S-V-A-2)*** *id = v;*, $H, V \longrightarrow \_, H, V, \Gamma \vdash \_ : T_1 \times T_2$ ***(Underscore Lemma)***.Since $E(id) = t$ and $\Gamma \vdash H, V$ $H(V(id)) = v'$ and $\Gamma \vdash v' : t$. Since $H' = H[v/l], H(V(id)) = v$ and $\Gamma \vdash v : t$, so $\Gamma \vdash H', V$.

3. ***(S-V-A-3)*** *id = ref{id', l};*, $H, V \longrightarrow \_, H, V, \Gamma \vdash \_ : T_1 \times T_2$ ***(Underscore Lemma)***. Since $R(id) = t$ and $\Gamma \vdash H, V$, $V(id) = l'$, $H(l') = v$ and $\Gamma \vdash v : t$. By inversion of the typing rules $H(l) = v$ and $\Gamma \vdash v : t$, so $\Gamma \vdash H, V'$.

4. ***(S-V-A-3)*** *id = Error;*, $H, V \longrightarrow \_, H, V, \Gamma \vdash Error : T_1 \times T_2$ ***(TS-C-8)***.

Figure 106: Preservation 2 Part 2 B

Continuation 20

Case 3: $S_1 S_2$

$\Gamma \vdash S_1 S_2 \colon T_1 \times T_2$, by inversion of the rules, $\Gamma \vdash S_1 \colon T_3 \times T_4$, and $\Gamma \vdash S_2 \colon T_5 \times T_6$

Consider the possible reductions:

1. ***(S-S-1)*** $S_1 S_2, H, V \longrightarrow, H', V'$ $(S_1 \neq let\ id : t = v; S_1 \neq return\ e;)$

   $S_1, H, V \longrightarrow S_1', H', V'$ ***(Statement Reduction Lemma)***. So, by inductive assumption, $\Gamma \vdash S_1' : T_3 \times T_4$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash S_1' S_2 : T_1 \times T_2$.

2. ***(S-S-2)*** $\_\ S_2, H, V \longrightarrow S_2, H, V$, $\Gamma \vdash\ S_2 : T_1 \times T_2$ ***(TS-C-5)***

3. ***(S-S-3)*** $return\ v; S_2, H, V \longrightarrow return\ v;, H, V : \Gamma \vdash return\ v; : T_1 \times T_2$. (Either return is the same type as the original expression, or ***(TS-C-7)*** is applicable).

4. ***(S-S-4)*** $Error\ S_2, H, V \longrightarrow Error, H, V$, $\Gamma \vdash\ Error : T_1 \times T_2$ ***(TS-C-8)***

Case 4: $return\ e;\ \Gamma \vdash return\ e; : T_1 \times T_2$, by inversion of the rules $\Gamma \vdash e : T_e$

1. ***(S-R-1)*** $return\ e;, H, V \longrightarrow return\ e';, H', V'$

   $e, H, V \longrightarrow e', H', V'$. So, by inductive assumption $\Gamma \vdash e : T_e$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash return\ e'; : T_1 \times T_2$.

2. ***(S-R-2)*** return Error;, $H, V \longrightarrow Error, H, V$. $\Gamma \vdash Error : T_1 \times T_2$ **(TS-C-8)**.

Case 5: $\Gamma \vdash if\ b\ \{S_1\}\ else\ \{S_2\} : T_1 \times T_2$. By inversion of the typing rule $\Gamma \vdash b : T_e$, $\Gamma \vdash S_1 : T_3 \times T_4$, and $\Gamma \vdash S_2 : T_5 \times T_6$

1. ***(S-C-1)*** $if\ b\ \{S_1\}\ else\ \{S_2\}, H, V \longrightarrow if\ b'\ \{S_1\}\ else\ \{S_2\}, H', V'$

   $b, H, V \longrightarrow b', H', V'$, so by inductive assumption $\Gamma \vdash b' : T_e$ and $\Gamma \vdash H', V'$. Thus, $\Gamma \vdash if\ b'\ \{S_1\}\ else\ \{S_2\} : T_1 \times T_2$.

2. ***(S-C-2)*** $if\ true\ \{S_1\}\ else\ \{S_2\}, H, V \longrightarrow S_1, H, V$. (Either S_1 is the same type as the original expression, or **(TS-C-7)** is applicable).

3. ***(S-C-3)*** $if\ false\ \{S_1\}\ else\ \{S_2\}, H, V \longrightarrow S_1, H, V$. (Either S_1 is the same type as the original expression, or **(TS-C-7)** is applicable).

4. ***(S-C-4)*** $if\ Error\ \{S_1\}\ else\ \{S_2\}, H, V \longrightarrow Error, H, V$. $\Gamma \vdash Error : T_1 \times T_2$ ***(TS-C-8)***

Figure 107: Preservation 2 Part 2 C

Continuation 21

Case 6: $\Gamma \vdash while\ b\ \{S_1\} : T_1 \times T_2$

1. ***(S-C-6)*** $while\ b\ \{St\}, H, V \longrightarrow if\ b\ \{St\ while\ b\ \{St\}\}\ else\{S_t\},\ H, V$

   Since they share the same condition b and the same statement St,
   $\Gamma \vdash if\ b\ \{St\ while\ b\ \{St\}\}\ else\{St\} : T_1 \times T_2$.

Case 7: $\Gamma \vdash e_1.id = e_2 : T \times void$, $\Gamma \vdash e_1 : t_1, \Gamma \vdash e_2 : t_2$

1. ***(S-S-A-1)*** $e_1.id = e_2, H, V \longrightarrow e_1'.id = e_2, H', V'$

   $e_1, H, V \longrightarrow e_1', H', V'$, by inductive assumption, $\Gamma \vdash e_1' : t_1$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash e_1'.id = e_2 : T \times void$.

2. ***(S-S-A-2)*** $*ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e, H, V \longrightarrow *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e', H', V'$

   $e_2, H, V \longrightarrow e_2', H', V'$, by inductive assumption, $\Gamma \vdash e_2' : t_2$ and $\Gamma \vdash H', V'$. So, $\Gamma \vdash *ref(id', S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id = e' : T \times void$.

3. ***(S-S-A-3)***$*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = v, H, V \longrightarrow \_, H', V'$

   $\Gamma \vdash \_ : T \times void$**(Underscore Lemma)**. Since $\Gamma \vdash H, H(l_i) = v'$ and $\Gamma \vdash v' : t_i$. By inversion of the typing rules, $H(l) = v$ and $\Gamma \vdash v : t_i$. $V' = V[id/l]$, so $\Gamma \vdash H', V'$.

4. ***(S-S-A-4)*** $*ref(id, S_{id}\{a_1 : t_1 = l_1, a_2 : t_2 = l_2, ...a_n : t_n = l_n\}).id' = ref\{id'', l\}, H, V \longrightarrow \_, H', V'$

   $\Gamma \vdash \_ : T \times void$ **(Underscore Lemma)**. Since $\Gamma \vdash H$, $H(l_i) = v'$ and $\Gamma \vdash v' : t_i$. By inversion of the typing rules $\Gamma \vdash v : t_i$, also $H' = H[v/l]$, so $\Gamma \vdash H', V'$

5. ***(S-S-A-4)*** $v.id' = v, H, V \longrightarrow Error, H, V$ $\Gamma \vdash Error : T \times void$, **(TS-C-8)**.

Case 8: $\Gamma \vdash$ **Global Definitions** St

1. ***(S-G-1)*** **Global Definitions** $St \longrightarrow St'$ **(Substitution Assumption)**

Case 9: $\Gamma \vdash e; : \alpha \times void$. By inversion of the rules, $\Gamma \vdash e : t$

1. ***(S-O-1)*** $e;, H \longrightarrow e';, H', V'$: By inductive assumption, $\Gamma \vdash e : t$, and $\Gamma \vdash H', V'$, so $\Gamma \vdash e; : \alpha \times void$.

2. $v;, H, V \longrightarrow \_, H, V$ $\Gamma \vdash \_ : \alpha \times void$.***(Underscore Lemma)***

3. $Error;, H, V \longrightarrow Error, H, V$ $\Gamma \vdash Error : \alpha \times void$. ***(TS-C-8)***

Figure 108: Preservation 2 Part 2 D

Continuation 22

Case 10: $\Gamma \vdash free(id);: \alpha \times void$

1. ***(S-O-4)*** $free(id);, H, V \longrightarrow \_, H', V'$ Since $id \notin R$, $\Gamma \vdash H', V'$.
2. ***(S-O-5)*** $free(id);, H, V \longrightarrow Error, H, V$. $\Gamma \vdash Error : \alpha \times void$ ***(TS-C-8)***

Figure 109: Preservation 2 Part 2 E

9 Concluding Theorems

Theorem 9.1 Safe Expression Soundness

If $\vdash E$: t and t is safe, E is an integer, a boolean, a reference or structure literal, or $E, H, V \longrightarrow E', H', V'$, and $\vdash E'$ and $\vdash H', V'$.

Proof: by **(Progress 1)** (Part 1), if $\vdash E$: t and E is safe, E is an integer, a boolean, a reference or structure literal, or $E, H, V \longrightarrow E', H', V'$. By **(Preservation 1)** (Part 1) $\vdash E' : t$ and $\vdash H', V'$.

Theorem 9.2 Safe Statement Soundness

If $\vdash$ E : $void \times T$, then E is _, or there exists some E', such that $E, H, V \longrightarrow E', H', V'$, and $\vdash E : void \times T$.

Proof: by **(Progress 1)** (Part 2), if $\vdash$ E : $void \times T$, then E is _, or there exists some E', such that $E, H, V \longrightarrow E', H', V'$. By **(Preservation 1)** (Part 2) $\vdash E' : void \times T$ and $\vdash H', V'$.

Theorem 9.3 Unsafe Expression Soundness

If $\vdash E$: t and t is unsafe, E is an integer, a boolean, a reference or structure literal, Error, or $E, H, V \longrightarrow E', H', V'$, and $\vdash E'$ and $\vdash H', V'$.

Proof: by **(Progress 1)** (Part 1), if $\vdash E$: t and E is safe, E is an integer, a boolean, a reference or structure literal, Error, or $E, H, V \longrightarrow E', H', V'$. By **(Preservation 1)** (Part 1) $\vdash E' : t$ and $\vdash H', V'$.

Theorem 9.4 Unsafe Statement Soundness

If $\vdash$ E : $\alpha \times T$, then E is _, Error, or there exists some E', such that $E, H, V \longrightarrow E', H', V'$.

Proof: by **(Progress 1)** (Part 2), if $\vdash$ E : $\alpha \times T$, then E is _, or there exists some E', such that $E, H, V \longrightarrow E', H', V'$. By **(Preservation 1)** (Part 2) $\vdash E' : \alpha \times T$ and $\vdash H', V'$.

Figure 110: Theorems 2

Theorem 9.5 Soundness

$\vdash E : T$, where T is any type derivable by the typing rules, then E is an integer, boolean, reference or structure literal, Error, _ or $E, H, V \longrightarrow E', H', V'$ and $\vdash E : T$.

Proof: T is either some safe t, an unsafe t, $void \times T'$ or $\alpha \times T'$, so by **(Safe Expression Soundness)**,**(Safe Statement Soundness)(Safe Expression Soundness)** or **(Safe Statement Soundness)**, E is an integer, boolean, reference or structure literal, Error, _ or $E, H, V \longrightarrow E', H', V'$ and $\vdash E : T$

Theorem 9.6 Safety

If $\vdash E : t$ and t is safe or $\vdash E : void \times T$, then E is an integer, a boolean, a safe reference or structure literal, or $E, H, V \longrightarrow E', H', V'$ and $\vdash E : t$ or $\vdash E : void \times T$ (depending on the initial type of E) and $\vdash H', V'$.

Proof: If $\vdash E : t$, **(Soundness 1)**, and if $\vdash E : void \times T$, **(Soundness 2).**

Figure 111: Theorems 2

REFERENCES

"BACK TO THE BUILDING BLOCKS: A Path Toward Secure AND Measurable Software." (accessed: 03.5.2024). https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf.

Chandrasekhar, Boyapati, Alexandru Salcianu, William Beebee Jr, and Martin Rinard. "Ownership Types for Safe Region-Based Memory Management in Real-Time Java." *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003, 324–337.

"CS 164: Lecture Notes, Fall 2020." (accessed: 01.10.2024). https://inst.eecs.berkeley.edu/~cs164/fa20/lectures/.

Dan, Grossman, Morrisett Greg, Trevor Jim, Hicks Michael, Wang Yanling, and Cheney James. "Formal type soundness for Cyclone's region system," 2001.

———. "Region-Based Memory Management in Cyclone." *Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, 2008, 307–325.

"IMP: a simple imperative language." (accessed: 01.21.2024). https://groups.seas.harvard.edu/courses/cs152/2016sp/lectures/lec05-imp.pdf.

Millstein, Todd. "Simply-typed Lambda Calculus." (accessed: 01.20.2024). https://courses.cs.washington.edu/courses/csep505/03au/handouts/simply-typed.pdf.

Pierce, Benjamin C. "Advanced topics in types and programming languages." MIT press, 2004.

———. "Types and programming languages." MIT press, 2002.

Rastogi, Aseem, Cédric Fournet Nikhil Swamy, Gavin Bierman, and Panagiotis Vekris. "Safe and efficient gradual typing for TypeScript." *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015, 167–180.

Sergio, Maffeis, John C. Mitchell, and Ankur Taly. "An operational semantics for JavaScript." *Programming Languages and Systems: 6th Asian Symposium, APLAS 2008*, 2008, 307–325.