C♭ AST Validation

1 Naming Rules

- No duplicate names amongst globals, externs, and functions.

- No duplicate names amongst struct types, and for a given struct type no duplicate names amongst its fields.

- For a given function, no duplicate names amongst its parameters and no duplicate names amongst its locals.

2 Miscellaneous Rules

- There exists a function called `main` and its type is $() \rightarrow$ `int`.

- For every function, all paths from that function's entry reach a `Return` statement.

3 Type Checking

3.1 Types

$\tau \in \mathit{Type} ::= $ `int` $\mid$ `struct`$_{Id}$ $\mid$ $\&\tau$ $\mid$ $(\vec{\tau}) \rightarrow \tau^?$

A type τ is either the integer type, a struct type (qualified by the name of the struct), a pointer type (qualified by the type being pointed to), or a function type (qualified by the types of the parameters and an optional return type). Examples:

- An integer value has type `int`.

- A struct named *foo* has type `struct`$_{foo}$.

- A pointer to an int has type $\&$`int`.

- A function taking an int and a pointer to an int and returning an int has type $(\text{`int`}, \&\text{`int`}) \rightarrow$ `int`.

- A function taking an int and a pointer to an int and not returning anything has type $(\text{`int`}, \&\text{`int`}) \rightarrow \_$.

3.2 Type Evironment

Let $\Gamma : Id \rightarrow \mathit{Type}$ be a map from identifiers (variable and function names) to types.

Let $\Delta : Id \rightarrow (Id \rightarrow \mathit{Type})$ be a map from identifers (struct names) to a map from identifiers (field names) to types.

We initialize Γ_0 and Δ from the the AST `Program` node's `globals`, `structs`, `externs`, and `functions` fields. Note that for a function inside `externs` the function name is mapped to its function type, while for a function inside `functions` the function name is mapped to a *pointer* to the function type. Also, we don't include the `main` function in Γ_0. At this point, Γ_0 has all the type information for global variables, extern declared functions, and internal defined functions (except `main`) and Δ has the typing information for all struct declarations.

Δ will stay constant throughout the type checking process. Γ_0 will be augmented when type checking each function's body to contain that function's parameters and locals.

3.3 Exp and Lval Typing Rules

Typing judgements for `Exp` and `Lval` nodes in the AST are of the form $\Gamma, \Delta \vdash e : \tau$, stating that given the type environment consisting of Γ and Δ, the AST node has type τ.

$$\frac{}{\Gamma, \Delta \vdash \texttt{Num(n)} : \texttt{int}} \text{ NUM}$$

$$\frac{\Gamma(\texttt{name}) = \tau}{\Gamma, \Delta \vdash \texttt{Id(name)} : \tau} \text{ ID}$$

$$\frac{}{\Gamma, \Delta \vdash \texttt{Nil} : \&\_} \text{ NIL}$$

We're using a convenient ad-hoc notation for the NIL rule expressing that a `Nil` expression is a pointer, but it can be treated as a pointer to any type. When comparing two types to see whether they are equal, `Nil`'s type should be considered equal to any other pointer type.

$$\frac{\Gamma, \Delta \vdash e : \texttt{int}}{\Gamma, \Delta \vdash \texttt{Unop(Neg, e)} : \texttt{int}} \text{ NEG}$$

$$\frac{\Gamma, \Delta \vdash e : \&\tau}{\Gamma, \Delta \vdash \texttt{Unop(Deref, e)} : \tau} \text{ DEREF}$$

$$\frac{\texttt{op} \in \{\texttt{Equal}, \texttt{NotEq}\} \qquad \Gamma, \Delta \vdash \texttt{left} : \tau \qquad \Gamma, \Delta \vdash \texttt{right} : \tau \qquad \tau \in \{\texttt{int}, \&\tau'\}}{\Gamma, \Delta \vdash \texttt{Binop(op, left, right)} : \texttt{int}} \text{ BINOP-EQ}$$

$$\frac{\texttt{op} \notin \{\texttt{Equal}, \texttt{NotEq}\} \qquad \Gamma, \Delta \vdash \texttt{left} : \texttt{int} \qquad \Gamma, \Delta \vdash \texttt{right} : \texttt{int}}{\Gamma, \Delta \vdash \texttt{Binop(op, left, right)} : \texttt{int}} \text{ BINOP-REST}$$

$$\frac{\Gamma, \Delta \vdash \texttt{ptr} : \&\tau \qquad \Gamma, \Delta \vdash \texttt{idx} : \texttt{int}}{\Gamma, \Delta \vdash \texttt{ArrayAccess(ptr, idx)} : \tau} \text{ ARRAY}$$

$$\frac{\Gamma, \Delta \vdash \texttt{ptr} : \&\texttt{struct}_{id} \qquad \Delta(id)(\texttt{fld}) = \tau}{\Gamma, \Delta \vdash \texttt{FieldAccess(ptr, fld)} : \tau} \text{ FIELD}$$

$$\frac{\texttt{callee} \neq \texttt{main} \qquad \Gamma, \Delta \vdash \texttt{callee} : \&(\vec{\tau}) \to \tau_r \qquad \forall (e, \tau) \in (\texttt{args}, \vec{\tau}) . [\Gamma, \Delta \vdash e : \tau]}{\Gamma, \Delta \vdash \texttt{Call(callee, args)} : \tau_r} \text{ CALL-INTERNAL}$$

$$\frac{\Gamma,\Delta \vdash \texttt{callee} : (\vec{\tau}) \to \tau_r \qquad \forall(\texttt{e},\tau) \in (\texttt{args},\vec{\tau}) . [\Gamma,\Delta \vdash \texttt{e} : \tau]}{\Gamma,\Delta \vdash \texttt{Call(callee, args)} : \tau_r} \text{ CALL-EXTERN}$$

We need two different call rules because internally defined functions are present in Γ as pointer types (in order to allow for function pointers and indirect calls) while externally defined functions are present in Γ as function types (because we can't call them indirectly via pointers).

3.4   Statement Typing Rules

Typing judgements for statement nodes in the AST are of the form $\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{stmt} : \mathbf{ok}$, where $\tau_r^?$ is the (optional) return type of the function containing $\texttt{stmt}$ and $\mathbf{loop}$ is a boolean indicating whether $\texttt{stmt}$ is contained inside a while loop.

$$\frac{\mathbf{loop} = \mathbf{true}}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{Break} : \mathbf{ok}} \text{ BREAK}$$

$$\frac{\mathbf{loop} = \mathbf{true}}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{Continue} : \mathbf{ok}} \text{ CONTINUE}$$

$$\frac{\tau_r^? = \mathbf{none}}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{Return(none)} : \mathbf{ok}} \text{ RETURN-1}$$

$$\frac{\Gamma,\Delta \vdash \texttt{e} : \tau \qquad \tau_r^? = \tau}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{Return(e)} : \mathbf{ok}} \text{ RETURN-2}$$

$$\frac{\Gamma,\Delta \vdash \texttt{lhs} : \tau \qquad \Gamma,\Delta \vdash \texttt{e} : \tau \qquad \tau \notin \{\mathbf{struct}_{id}, (\vec{\tau}) \to \tau_r\}}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{Assign(lhs, e)} : \mathbf{ok}} \text{ ASSIGN-EXP}$$

$$\frac{\Gamma,\Delta \vdash \texttt{lhs} : \&\tau \qquad \Gamma,\Delta \vdash \texttt{e} : \texttt{int} \qquad \tau \neq (\vec{\tau}) \to \tau_r}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{Assign(lhs, New(τ, e))} : \mathbf{ok}} \text{ ASSIGN-NEW}$$

$$\frac{\texttt{callee} \neq \texttt{main} \qquad \Gamma,\Delta \vdash \texttt{callee} : \&(\vec{\tau}) \to \tau' \qquad \forall(\texttt{e},\tau) \in (\texttt{args},\vec{\tau}) . [\Gamma,\Delta \vdash \texttt{e} : \tau]}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{Call(callee, args)} : \mathbf{ok}} \text{ CALL-INTERNAL}$$

$$\frac{\Gamma,\Delta \vdash \texttt{callee} : (\vec{\tau}) \to \tau' \qquad \forall(\texttt{e},\tau) \in (\texttt{args},\vec{\tau}) . [\Gamma,\Delta \vdash \texttt{e} : \tau]}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{Call(callee, args)} : \mathbf{ok}} \text{ CALL-EXTERN}$$

$$\frac{\Gamma,\Delta \vdash \texttt{e} : \texttt{int} \qquad \forall s \in \texttt{tt} . [\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash s : \mathbf{ok}] \qquad \forall s \in \texttt{ff} . [\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash s : \mathbf{ok}]}{\Gamma,\Delta,\tau_r^?,\mathbf{loop} \vdash \texttt{If(e, tt, ff)} : \mathbf{ok}} \text{ IF}$$

$$\frac{\Gamma, \Delta \vdash \mathtt{e} : \mathtt{int} \qquad \forall s \in \mathtt{body} \,.\, \Gamma, \Delta, \tau_r^?, \mathbf{true} \vdash s : \mathbf{ok}}{\Gamma, \Delta, \tau_r^?, \mathbf{loop} \vdash \mathtt{While(e, body)} : \mathbf{ok}} \; \text{WHILE}$$

3.5   Program Typing Rules

Typing judgements for the `Program` node and its children in the AST are of the form $\Gamma, \Delta \vdash \mathtt{node} : \mathbf{ok}$.

$$\frac{\forall g \in \mathtt{globs} \,.\, [\Gamma_0, \Delta \vdash g : \mathbf{ok}] \qquad \forall s \in \mathtt{structs} \,.\, [\Gamma_0, \Delta \vdash s : \mathbf{ok}] \qquad \forall f \in \mathtt{funcs} \,.\, [\Gamma_0, \Delta \vdash f : \mathbf{ok}]}{\Gamma_0, \Delta \vdash \mathtt{Program(globs, structs, externs, funcs)} : \mathbf{ok}} \; \text{PROGRAM}$$

The PROGRAM rule is the entry point for the type checker; it kicks everything off by calling all the other rules directly or transitively. Note that it explicitly uses Γ_0, the initial type environment, whereas all other rules use whatever type environment is passed to them.

$$\frac{\tau \notin \{\mathbf{struct}_{id}, (\vec{\tau}) \to \tau_r\}}{\Gamma, \Delta \vdash \mathtt{Decl(name, \tau)} : \mathbf{ok}} \; \text{GLOBAL}$$

$$\frac{\forall \mathtt{Decl(name, \tau)} \in \mathtt{flds} \,.\, [\tau \notin \{\mathbf{struct}_{id}, (\vec{\tau}) \to \tau_r\}]}{\Gamma, \Delta \vdash \mathtt{Struct(name, flds)} : \mathbf{ok}} \; \text{STRUCT}$$

$$\frac{\begin{array}{c} \Gamma' = \mathtt{prms} \sqcup \mathtt{locals.decls} \qquad \forall \mathtt{Decl(name, \tau)} \in \Gamma' \,.\, [\tau \notin \{\mathbf{struct}_{id}, (\vec{\tau}) \to \tau_r\}] \qquad \Gamma'' = \Gamma \sqcup \Gamma' \\ \forall (\mathtt{Decl(name, \tau), e}) \in \mathtt{locals} \,.\, [\Gamma'', \Delta \vdash \mathtt{e} : \tau] \qquad \forall s \in \mathtt{stmts} \,.\, [\Gamma'', \Delta, \tau_r^?, \mathbf{false} \vdash s : \mathbf{ok}] \end{array}}{\Gamma'', \Delta \vdash \mathtt{Function(name, prms, \tau_r^?, locals, stmts)} : \mathbf{ok}} \; \text{FUNCTION}$$

The notation $\Gamma' = \mathtt{prms} \sqcup \mathtt{locals.decls}$ means to turn the parameter and local declarations (ignoring any optional local initializations) into a type environment, i.e., a map from names to types, allowing any entries from the right-hand side to override entries with the same name on the left-hand side. In other words, a local declaration overrides a parameter declaration with the same name. Similarly for $\Gamma'' = \Gamma \sqcup \Gamma'$ any entry from Γ' overrides an entry with the same name in Γ.