Package Type System for Sketch

Santiago Gonzalez

Summer 2014

1 Type Rules

The following rules ignore global field declarations. They are always private.

$$Ancestors(P) \quad = \quad \{P_A | P \text{ impl } P_A \vee (\exists P_B \; P \text{ impl } P_B \wedge P_A \in Ancestor(P_B))\}$$

Typechecking program (root)

$$\frac{\forall P\{F;S\} \in Prog \; Prog \vdash P}{\Gamma \vdash Prog}$$

Interfaces for package hierarchies [1]

$$\frac{S_I \subseteq S_P \quad F_I \subseteq F_P \quad \forall \{P_C\{F_C;S_C\} | P_C \text{ impl } P_p\} \Rightarrow I[\{F_C;S_C\}/\{F_P;S_P\}] \sqsubseteq P_C}{I\{F_I;S_I\} \sqsubseteq P_P\{F_P;S_P\}}$$

Typechecking packages

$$\frac{\forall f \in F \; (Prog,P) \vdash f \quad \forall s \in S \; (Prog,P) \vdash s}{Prog \vdash P\{F;S\}}$$

Typechecking functions

$$\frac{(Prog,P) \vdash b \quad (Prog,P) \vdash \tau \quad \forall \tau_a \in args \; (Prog,P) \vdash \tau_a}{(Prog,P) \vdash \tau \text{ name}(args) \; \{b\}}$$

[1] Note: when we say a function is an element of a function set, we use its signature (i.e. the function's name, return type, and arguments) to differentiate it, not just its name. However, two different return types / argument types, τ_1 and τ_2, from otherwise same signatured functions, f_1 and f_2, should be considered the same if f_2's containing package is a subpackage of f_1's containing package (i.e. f_2 is a more *specific* implementation of f_1) and τ_2's containing package is a subpackage of τ_1's containing package.

Typechecking function call (explicit package)

$$\frac{Q\{F_Q; S_Q\} \quad P\{F_P; S_P\} \quad Q \neq P \quad I\{F_I; S_I\} \sqsubseteq Q \quad Q \notin An.(P) \quad P \notin An.(Q) \quad foo \in F_I}{(Prog, P) \vdash \text{foo@Q}()}$$

Typechecking function call (from unspecified package)

$$\frac{P\{F_P; S_P\} \quad foo \notin F_P \quad \exists Q, foo \in F_Q \quad \forall M \neq Q, foo \notin F_M \quad I\{F_I; S_I\} \sqsubseteq Q \quad foo \in F_I}{(Prog, P) \vdash \text{foo}()}$$

Typechecking function call (from within same package) `// takes precedence over unspecified package rule`

$$\frac{P\{F_P; S_P\} \quad foo \in F_P}{(Prog, P) \vdash \text{foo}()}$$

Typechecking struct use (from within same package) `// takes precedence over unspecified package rule`

$$\frac{P\{F_P; S_P\} \quad Bar \in S_P}{(Prog, P) \vdash \text{Bar b} = \cdots}$$

Typechecking struct use (from unspecified package)

$$\frac{P\{F_P; S_P\} \quad Bar \notin S_P \quad \exists Q, Bar \in S_Q \quad \forall M \neq Q, Bar \notin S_M \quad I\{F_I; S_I\} \sqsubseteq Q \quad Bar \in S_I}{(Prog, P) \vdash \text{Bar b} = \cdots}$$

Typechecking struct use (explicit package)

$$\frac{Q\{F_Q; S_Q\} \quad P\{F_P; S_P\} \quad Q \neq P \quad I\{F_I; S_I\} \sqsubseteq Q \quad Q \notin An.(P) \quad P \notin An.(Q) \quad Bar \in S_I}{(Prog, P) \vdash \text{Bar@Q b} = \cdots}$$

Typechecking struct field access (to allow utility packages with fully public structs)

$$\frac{Q\{F_Q; S_Q\} \quad P\{F_P; S_P\} \quad Q \neq P \quad st : \tau \quad \tau \in S_Q \quad \forall M \neq Q, !(M \text{ impl } Q) \wedge !(Q \text{ impl } M)}{(Prog, P) \vdash \text{st.field}}$$

Typechecking struct field access (from within same package)

$$\frac{P\{F_P; S_P\} \quad st : t \quad \tau \in S_P}{(Prog, P) \vdash \text{st.field}}$$

1.1 Reasoning for Conservative Function/Struct Usage Allowances

We want to avoid situations where the removal of a package that is not being used causes something else to not work. For example, consider a program with package A, B, and C where packages B and C are implementations of A. These packages are all in separate files. The program's main file contains a method where a certain function in B is used and this

file contains includes for A, B, and C. The removal of the include for C would modify certain behaviors (i.e. private/publicness of certain functions) that would cause the program to behave incorrectly without the programmer's knowledge. The inclusion of C introduces new constraints into such behaviors that removes ambiguities. These conservative usage allowances of functions/structs, from unspecified packages, are in place to prevent such situations.