Rhyme Prototype Type Rules

November 10, 2024

Contents

1 Primitive Types

Firstly, there are two types named "Any" (denoted $\top$) and "Never" (defined $\bot$). Any is a supertype of all types, while Never is a subtype of all types. When considering types as sets of possible values, it can be helpful to consider Any as the set of all possibles values, and Never as the empty set.

For numbers, there are 8, 16, 32, and 64 bit variants of signed and unsigned integers, aswell at 32 and 64bit floating point integers.

For these number values:

Let $U = \{u8, u16, u32, u64\}$. Let $I = \{i8, i16, i32, i64\}$, and $F = \{f32, f64\}$.

Let $N = U \cup I \cup F$.

For strings, there is a base string type, String, and constant strings. In terms of sets, String can be thought of as the set of all strings, while a constant string is a set containing a singular, constant string.

There are aswell more types: Keys, Objects, and Functions, that are defined later, aswell as an Unknown type.

2 Subtype Definition and Properties

Consider the following as the definition of a type being a subtype.

$$\frac{\Gamma \vdash x : T \quad \vdash T \leq: S}{\Gamma \vdash x : S}$$

$$\frac{}{\vdash T \leq: T}$$

$$\frac{\vdash T \leq: S \quad \vdash S \leq: T}{\vdash S =: T}$$

Let the following be the definition of the =: relation between types.

$$\frac{\vdash S =: T}{\vdash T \leq: S \quad \vdash S \leq: T}$$

2.1 Subtype Relationships

By definition of Any and Never, the following must be true, for any type T:

$$\frac{}{\vdash T \leq: \top}$$

$$\frac{}{\vdash \bot \leq: T}$$

2.1.1 Numbers

$u8 \leq: u16 \leq: u32 \leq: u64$

$i8 \leq: i16 \leq: i32 \leq: i64$

$f32 \leq: f64$, $u8 \leq: i16$, $u16 \leq: i32$, $u32 \leq: i64$

Technically, these following subtypes can also be added, given the size of the mantissa of floats. However, given CPU architecture design, a conversion would be required, which would add complexity. It might be better to consider an implicit/explicit casting operation.

$i16 \leq: f32$, $u16 \leq: f32$

$i32 \leq: f64$, $u32 \leq: f64$

2.1.2 Strings

$$\frac{\vdash S \text{ is a constant string}}{\vdash S \leq: \text{string}}$$

3 Union Definition and Properties

$$\frac{}{\vdash T \leq: (T \cup S)}$$

$$\frac{}{\vdash (S \cup T) =: (T \cup S)}$$

$$\frac{}{\vdash (T \cup T) =: T}$$

$$\frac{\vdash T \leq: S}{\vdash (T \cup S) =: S}$$

4 Intersection Definition and Properties

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash t : S}{\Gamma \vdash t : (T \cap S)}$$

$$\frac{}{\vdash (T \cap S) \leq: T}$$

$$\frac{}{\vdash (S \cap T) =: (T \cap S)}$$

$$\frac{}{\vdash (T \cap T) =: T}$$

$$\frac{\vdash T \leq: S}{\vdash (T \cap S) =: T}$$

$$\frac{\vdash T' \leq: T \quad \vdash S \cap T' =: P \quad \vdash S \cap (T \setminus T') =: Q}{\vdash S \cap T =: P \cup Q}$$

4.1 Non-Intersecting Types

$$\frac{}{\vdash T \cap \bot =: \bot}$$

$$\frac{\vdash T \leq: \text{u64} \cup \text{i64} \cup \text{f64} \quad \vdash S \leq: \text{string}}{\vdash T \cap S =: \bot}$$

$$\frac{\vdash T \leq: \text{u64} \cup \text{i64} \cup \text{f64} \quad \vdash S \leq: \{\}}{\vdash T \cap S =: \bot}$$

$$\frac{\vdash T \leq: \{\} \quad \vdash S \leq: \text{string}}{\vdash T \cap S =: \bot}$$

$$\frac{\vdash T \text{ is a constant string} \quad \vdash S \text{ is a constant string} \quad \vdash T \neq S}{\vdash T \cap S =: \bot}$$

5 NotEmpty Types

With intersections constricting types to a smaller set than either input could yield, it can happen that the intersection of two types that are both nonempty yields a Never type. This has serious problems whenever it comes to filter variables, and specifically object accesses using them, because there is no guarantee that two keys in objects will appropriately overlap and their intersection be nonempty. However, there are certain compile time guarantees that can be made if a type is known for certain to be nonempty. As such, it's important to define a way of determining if a type can be guranteed to be known to be nonempty at compiletime. Aswell, implementation-wise, the programmer can explicitly tell the type system if a type is nonempty as hints, .

As such, the NotEmpty function is defined as follows:

$$\frac{\vdash S <: T \quad \vdash \text{NotEmpty}(S)}{\vdash \text{NotEmpty}(T)}$$

$$\frac{\vdash T \in N}{\vdash \text{NotEmpty}(T)}$$

$$\frac{}{\vdash \text{NotEmpty}(\text{String})}$$

$$\frac{\vdash S \text{ is a constant string}}{\vdash \text{NotEmpty}(S)}$$

$$\frac{}{\vdash \text{NotEmpty}(\{\})}$$

$$\frac{}{\vdash \text{NotEmpty}(S\{K : V\})}$$

6 Result Propagation

It is possible that the evaluation of an expression does not yield any values, or it throws an error. As such, there are two properties: the Nothing ($\mathcal{N}$) and Error ($\mathcal{E}$) properties. If an expression t yields an $i64$ type, and can return nothing, then it is written as $\Gamma \vdash t : i64 \mid \mathcal{N}$.

Nothing and Error results inherently should be propagated whenever possible, except for in certain, usually stateful, expressions, that remove the possibility of nothing by assuming a default value.

7 Number Types

7.1 Pure Operations on Numbers

Given the subtype relations, the following rules work for most instances, such as u8 + i16:

$$\frac{\vdash T \in N \quad \Gamma \vdash a : T \mid P_1 \quad \Gamma \vdash b : T \mid P_2}{\Gamma \vdash a + b : T \mid P_1 \cup P_2}$$

These rules can then be extended to most other pure (math) operations.

TODO: Determine how to possibly handle errors of division by zero or similar. Perhaps explicitly include the possibility of returning Nothing, as follows:

$$\frac{\vdash T \in N \quad \Gamma \vdash a : T \mid P_1 \quad \Gamma \vdash b : T \mid P_2}{\Gamma \vdash a/b : T \mid P_1 \cup P_2 \cup \mathcal{N}}$$

TODO: Figure out relations between floats and integers, aswell as unsigned and signed integers. Specifically,

when they're the same type.

7.2 Stateful Operations on Numbers

Same TODO applies for the relation between floats, signed ints, and unsigned ints, outside of the subtypes given.

$$\frac{\vdash T \in N \quad \Gamma \vdash a : T \mid P \cup \mathcal{N}}{\Gamma \vdash \text{sum}(a) : T \mid P}$$

$$\frac{\vdash T \in N \quad \Gamma \vdash a : T \mid P \cup \mathcal{N}}{\Gamma \vdash \text{product}(a) : T \mid P}$$

$$\frac{\vdash T \in N \quad \Gamma \vdash a : T \mid P \cup \mathcal{N}}{\Gamma \vdash \text{sum}(a) : T \mid P}$$

$$\frac{\vdash T \in N \quad \Gamma \vdash a : T \mid P \cup \mathcal{N}}{\Gamma \vdash \text{product}(a) : T \mid P}$$

$$\frac{\vdash T \in N \quad \Gamma \vdash a : T \mid P}{\Gamma \vdash \min(a) : T \mid P}$$

$$\frac{\vdash T \in N \quad \Gamma \vdash a : T \mid P}{\Gamma \vdash \max(a) : T \mid P}$$

8 Key Types

Let there be a "key" type $K_n(T)$ defined by $\frac{}{\vdash K_n(T) \leq: T}$.

$$\frac{}{\vdash K_n(T) \leq: T}$$

$$\frac{\vdash T_1 \cap T_2 =: \bot}{\vdash K_{n_1}(T_1) \cap K_{n_2}(T_2) =: \bot}$$

8.1 Pure Operations on Keys

Given $x : K_n(T)$, we cannot guarantee that $x + y : K_n(T)$. Hence, plus, minus, etc must yield type T. As such, these rules are then covered since $K_n(T) \leq: T$, and previous rules cover the case of $x : T$

8.2 Stateful Operations on Keys

In a similar vein to the pure operations for keys, sum and product cannot guarantee the result would be a key, so they must return the key's supertype. Hence, the previous rules aswell cover sum and product.

$$\frac{T \in N \quad \vdash a : K_n(T) \mid P}{\vdash \min(a) : K_n(T) \mid P}$$

$$\frac{T \in N \quad \vdash a : K_n(T) \mid P}{\vdash \max(a) : K_n(T) \mid P}$$

9 Object Types

Objects are defined recursively through a base object alongside updates that insert a key value type pairs into the object. While K is used as the variable for an object key, K does not necessarily have to be a key

type $K_n(T)$.

$$\frac{}{\Gamma \vdash \{\} : \{\} \mid \emptyset}$$

$$\frac{\Gamma \vdash x : T \mid P \quad \Gamma \vdash k : K \mid P_1 \quad \Gamma \vdash v : V \mid P_2}{\Gamma \vdash x\{k : v\} : T\{K_n(K) : V\} \mid P \cup ((P_1 \cup P_2) \setminus \mathcal{N})}$$

$$\frac{\Gamma \vdash x : T \mid P \quad \Gamma \vdash k : K \mid P_1 \setminus \mathcal{N} \quad \Gamma \vdash v : V \mid P_2 \setminus \mathcal{N}}{\Gamma \vdash x\{k : v\} : T\{K_n(K) : V\} \mid P \cup P_1 \cup P_2 \quad \vdash \text{NotEmpty}(K_n(K))}$$

10 Object Accessing

Given the recursive definition of an object, accessing the value of an object then recursively searches for potential values given a key type.

$$\frac{}{\Gamma \vdash \{\}[k] : \bot \mid \mathcal{N}}$$

$$\frac{\Gamma \vdash t : T\{K : V_1\} \mid P_1 \quad \Gamma \vdash k' : K' \mid P_2 \quad \vdash K' \leq: K}{\Gamma \vdash t[k'] : V_1 \mid P_1 \cup P_3}$$

$$\frac{\Gamma \vdash t : T\{K : V_1\} \mid P_1 \quad \Gamma \vdash k' : K' \mid P_2 \quad \Gamma, k' : (K' \setminus K \mid P_2) \vdash \text{parent}(t)[k'] : V_2 \mid P_3}{\Gamma \vdash t[k'] : V_1 \cup V_2 \mid P_1 \cup P_2 \cup P_3}$$

$$\frac{\Gamma \vdash t : T\{K : V_1\} \mid P_1 \quad \Gamma \vdash k' : K' \mid P_2 \quad \vdash K' \cap K =: \bot \quad \Gamma \vdash \text{parent}(t)[k'] : V_2 \mid P_3}{\Gamma \vdash t[k'] : V_2 \mid P_1 \cup P_2 \cup P_3}$$

Where $\text{parent}(t\{k : v\}) = t$

10.1 Determining types of filter variables

Given an object type, the domain of the type is given by the following function, denoted as the "keyof" function:

$\text{keyof}(\{\}) = \bot$

$\text{keyof}(t\{k : v\} : T\{K : V\}) = K \cup \text{keyof}(t)$

As for determining the types of filter variables, let p be the program being analyzed. The type of a filter variable f is defined as the fixpoint of $T = \text{ftype}^f_T[\![p]\!]$ with initial $T = \top$.

$\text{ftype}^f_T[\![\ c\ ]\!] = T$

$\text{ftype}^f_T[\![\ x\ ]\!] = T$

$\text{ftype}^f_T[\![\ e[f']\ ]\!] = \text{ftype}^f_T[\![\ e\ ]\!] \cap \text{keyof}(e)$ when $f = f'$

$\text{ftype}^f_T[\![\ e[f']?\ ]\!] = \text{ftype}^f_T[\![\ e\ ]\!]$

$\text{ftype}^f_T[\![\mathscr{F}\ e_1\ e_2]\!] = \text{ftype}^f_T[\![\ e_1\ ]\!] \cap \text{ftype}^f_T[\![\ e_2\ ]\!]$

$\text{ftype}^f_T[\![\mathcal{F}\ e]\!] = \text{ftype}^f_T[\![\ e\ ]\!]$

Because the result is always a product of intersections, and narrower intersections will only yield narrower results, this fixpoint is guaranteed to eventually converge onto some type. However, it is possible that type is $\bot$.

The type of the filter variable is then given by the inference rules:

$$\frac{\vdash \text{NotEmpty}(\text{fix}(\text{ftype}^f_T[\![\ p\ ]\!]))}{\Gamma \vdash f : \text{fix}(\text{ftype}^f_T[\![\ p\ ]\!]) \mid \emptyset}$$

$$\frac{}{\Gamma \vdash f : \text{fix}(\text{ftype}^f_T[\![\ p\ ]\!]) \mid \mathcal{N}}$$

where the initial value of T in the fixpoint is $\top$, and p is the entire program.

11 General Stateful Operations

Certain stateful operations don't depend on types.

$$\frac{\Gamma \vdash a : T \mid P}{\Gamma \vdash \text{first}(a) : T \mid P}$$

$$\frac{\Gamma \vdash a : T \mid P}{\Gamma \vdash \text{last}(a) : T \mid P}$$

$$\frac{\Gamma \vdash a : T \mid P}{\Gamma \vdash \text{single}(a) : T \mid P}$$

TODO: Determine size of objects. The following rules assume amount of items in object won't exceed maximum value of 32-bit unsigned integer.

$$\frac{\Gamma \vdash a : T \mid P \setminus \mathcal{N}}{\Gamma \vdash \text{array}(a) : \{\}\{K_n(u32) : T\} \mid P \quad \vdash \text{NotEmpty}(K_n(u32))}$$

$$\frac{\Gamma \vdash a : T \mid P \cup \mathcal{N}}{\Gamma \vdash \text{array}(a) : \{\}\{K_n(u32) : T\} \mid P}$$

12 Functions

12.1 Definition

$$\frac{\Gamma \vdash f : ((T_1, T_2, ..., T_n) \Rightarrow S) \mid P \quad \Gamma \vdash x_1 : T_1 \mid P_1, x_2 : T_2 \mid P_2, ..., x_n : T_n \mid P_n}{\Gamma \vdash f(x_1, x_2, ..., x_n) : S \mid P \cup P_1 \cup P_2 \cup ... \cup P_n}$$

12.2 Subtyping Relation

$$\frac{T_1 \leq: S_1, T_2 \leq: S_2, ..., T_n \leq: S_n \quad R_1 \leq: R_2}{(S_1, S_2, ..., S_n) \Rightarrow R_1 \leq: (T_1, T_2, ..., T_n) \Rightarrow R_2}$$

13 Unknown Type

Consider a new type Unknown.

A value with type of Unknown can be used in place of a value with a specific type T. However, in the instance that it is not of type T, then an error will be thrown. For certain operations in which no restrictions on

types are made, this means no error will be thrown. However, for operations with restrictions (e.g. addition can only occur with numbers), an error can be thrown.

Unknown is neither a supertype or subtype of any specific type.

13.1 Pure Operations on Unknown

Note: Cannot have $y : T \mid \emptyset \to x + y : T \mid \mathcal{E}$, because if T is a u8 and Unknown is a u64, then $x + y : u64$. It is necessary that $x + y$ hence is a supertype that can encapsulate all possible numbers. However, it is undecided as to how $u64$, $i64$, and $f64$ are to be related. As such, the union of these will be explicitly shown.

$$\frac{\vdash x : \text{Unknown} \mid P_1 \quad \vdash y : T \mid P_2 \quad \vdash T \in N}{\vdash x + y : u64 \cup i64 \cup f64 \mid P_1 \cup P_2 \cup \mathcal{E}}$$

$$\frac{\vdash x : \text{Unknown} \mid P_1 \quad \vdash y : \text{Unknown} \mid P_2}{\vdash x + y : u64 \cup i64 \cup f64 \mid P_1 \cup P_2 \cup \mathcal{E}}$$

13.2 Stateful Operations on Unknown

Stateful operations requiring numbers or comparable values would have to result in either a number or an Error.

$$\frac{\vdash x : \text{Unknown} \mid P}{\vdash \min(x) : u64 \cup i64 \cup f64 \mid P \cup \mathcal{E}}$$

$$\frac{\vdash x : \text{Unknown} \mid P}{\vdash \max(x) : u64 \cup i64 \cup f64 \mid P \cup \mathcal{E}}$$

$$\frac{\vdash x : \text{Unknown} \mid P \cup \mathcal{N}}{\vdash \text{sum}(x) : u64 \cup i64 \cup f64 \mid P \cup \mathcal{E}}$$

$$\frac{\vdash x : \text{Unknown} \mid P \cup \mathcal{N}}{\vdash \text{product}(x) : u64 \cup i64 \cup f64 \mid P \cup \mathcal{E}}$$

The other stateful operations covered in the general rules would apply without problem.

13.3 Object Access Operation with Unknown

There's two possible ways of handling object accesses on Unknown types. Given an unknown type could be a non-object, the result must inherently error. However, even if successful, the result type would still be unknown. Hence, it can be Unknown, or, alternatively, it can be Any. Of the two, propagating the existing type of Unknown appears to make more sense, hence the rule is:

$$\frac{\vdash x : \text{Unknown} \mid P_1 \quad \vdash k : T \mid P_2}{\vdash x[k] : \text{Unknown} \mid P_1 \cup P_2 \cup \mathcal{E}}$$

Considering the opposite then aswell, the existing rules for accessing should apply. Specifically note though that the intersection between unknown and the keys is non-empty, hence all possible values could be included as a result of the operation, aswell as Nothing.