Rhyme Prototype Type Rules

October 16, 2024

Contents

1 Subtype Definition and Properties

1.1.) $\dfrac{\vdash x : T \quad T \leq: S}{\vdash x : S}$

1.2.) $\dfrac{}{T \leq: T}$

1.3.) $\dfrac{T \leq: S \quad S \leq: T}{S =: T}$

1.4.) $\dfrac{S =: T}{T \leq: S \quad S \leq: T}$

Consider 1.1 as the definition of a type being a subtype.

Let 1.3 and 1.4 be the definition of the =: relation between types.

2 Union Definition and Properties

2.1.) $\dfrac{}{T \leq: (T \cup S)}$

2.2.) $\dfrac{}{(S \cup T) =: (T \cup S)}$

2.3.) $\dfrac{}{(T \cup T) =: T}$

2.4.) $\dfrac{T \leq: S}{(T \cup S) =: S}$

3 Primitive Types

For unsigned and signed integers, there are 8, 16, 32, and 64 bit variants. Aswell, there are 32bit and 64bit floating point types.

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.

There are aswell more types: Keys, Objects, and Functions, that are defined later.

3.1 Subtype Relationships

3.1.1 Numbers

3.1.1.1.) $u8 \leq: u16 \leq: u32 \leq: u64$

3.1.1.2.) $i8 \leq: i16 \leq: i32 \leq: i64$

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

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.

3.1.1.4.) $i16 \leq: f32$, $u16 \leq: f32$

3.1.1.5.) $i32 \leq: f64$, $u32 \leq: f64$

3.1.2 Strings

3.1.2.1.) $\dfrac{S \text{ is a constant string}}{S \leq: \text{string}}$

Also, just to include:

3.1.2.2.) $\dfrac{\vdash s = S \text{ where } S \text{ is a constant string}}{\vdash s : S}$

4 Number Types

4.1 Pure Operations on Numbers

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

4.1.1.) $\dfrac{T \in N \quad \vdash a : T \quad \vdash b : T}{\vdash a + b : T}$

4.1.2.) $\dfrac{T \in N \quad \vdash a : (T \cup \text{Nothing}) \quad \vdash b : (T \cup \text{Nothing})}{\vdash a + b : (T \cup \text{Nothing})}$

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 disclude the certainty of the result being T, as follows:?

4.1.3.) $\dfrac{T \in N \quad \vdash a : (T \cup \text{Nothing}) \quad \vdash b : (T \cup \text{Nothing})}{\vdash a/b : (T \cup \text{Nothing})}$

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

4.1.4.) $\dfrac{T_F \in F \quad T_I \in U \cup I \quad \vdash a : T_F \quad \vdash b : T_I}{\vdash a + b : T_F}$ (and it's commutative corrolary)

4.2 Stateful Operations on Numbers

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

4.2.1.) $\dfrac{T \in N \quad \vdash a : (T \cup \text{Nothing})}{\vdash \text{sum}(a) : T}$

4.2.2.) $\dfrac{T \in N \quad \vdash a : (T \cup \text{Nothing})}{\vdash \text{product}(a) : T}$

4.2.3.) $\dfrac{T \in N \quad \vdash a : (T \cup \text{Nothing})}{\vdash \min(a) : (T \cup \text{Nothing})}$

4.2.4.) $\dfrac{T \in N \quad \vdash a : (T \cup \text{Nothing})}{\vdash \max(a) : (T \cup \text{Nothing})}$

5 Key Types

Let $K_n(T)$ be a "key" type, defined by: $\dfrac{}{K_n(T) \leq: T}$.

5.1 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 $x : T$

5.2 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.

5.2.1.) $\dfrac{T \in N \quad \vdash a : (K_n(T) \cup \text{Nothing})}{\vdash \min(a) : (K_n(T) \cup \text{Nothing})}$

5.2.2.) $\dfrac{T \in N \quad \vdash a : (K_n(T) \cup \text{Nothing})}{\vdash \max(a) : (K_n(T) \cup \text{Nothing})}$

6 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 $K_n(T)$.

6.1.1.) $\dfrac{}{\vdash \{\} : \{\}}$

6.1.2.) $\dfrac{}{\{\} \in \text{object}}$

6.1.3.) $\dfrac{\vdash x : T \quad T \in \text{object}}{\vdash x\{K_1 : V_1\} : T\{K_1 : V_1\}}$

6.1.4.) $\dfrac{T \in \text{object}}{T\{K_1 : V_1\} \in \text{object}}$

7 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.

$$\frac{}{\{\}[k] : \text{Nothing}}$$

$$\frac{\vdash x : O \quad O \in \text{object} \quad \vdash k : K \quad K \leq: K_1}{\vdash (x\{K_1 : V_1\})[k] : V_1}$$

$$\frac{\vdash x : O \quad O \in \text{object} \quad \vdash k : K \quad K \cap K_1 \neq \emptyset \quad \vdash x[k : K \setminus K_1] : T_2}{\vdash (x\{K_1 : T_1\})[k] : T_1 \cup T_2}$$

$$\frac{\vdash x : O \quad O \in \text{object} \quad \vdash k : K \quad K \cap K_1 = \emptyset \quad \vdash x[k] : T_2}{\vdash (x\{K_1 : T_1\})[k] : T_2}$$

$$\frac{\vdash x : O \quad O \in \text{object} \quad k \text{ is a filter variable on } x}{\vdash k : \text{keyof}(O)}$$

where keyof is defined as

$$\frac{}{\text{keyof}(\{\}) =: \emptyset} \quad \frac{T \in \text{object}}{\text{keyof}(T\{K_1 : V_1\}) =: (K_1 \cup \text{keyof}(T))}$$

TODO: The base case of keyof implies the existence of a type with no possible value to it ($\emptyset$). Should this be allowed?

8 General Stateful Operations

Certain stateful operations don't depend on types.

$$\frac{\vdash a : (T \cup \text{Nothing})}{\vdash \text{first}(a) : (T \cup \text{Nothing})}$$

$$\frac{\vdash a : (T \cup \text{Nothing})}{\vdash \text{last}(a) : (T \cup \text{Nothing})}$$

$$\frac{\vdash a : (T \cup \text{Nothing})}{\vdash \text{single}(a) : (T \cup \text{Nothing})}$$

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

$$\frac{\vdash a : (T \cup \text{Nothing})}{\vdash \text{array}(a) : \{\}\{K_n(u32) : T\}}$$

$$\frac{\vdash a : T}{\vdash \text{count}(a) : u32}$$

9 Functions

9.1 Definition

$$\frac{\vdash f : (T_1, T_2, ..., T_n) \Rightarrow R \quad \vdash x_1 : T_1, x_2 : T_2, ..., x_n : T_n}{f(x_1, x_2, ..., x_n) : R}$$

9.2 Subtyping Relation

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

10 Unknown Type

Let there be two new types: Unknown and Error.

In general, if a variable x is of type Unknown, then if it being type T yields a result $f(x)$ of type S, then $f(x)$ should be of type $S \cup \text{Error}$.

TODO: Consider: Should Unknown be a subtype or supertype of anything? Does Unknown contain Nothing?

Because it is a question whether Unknown contains nothing, ($\text{Unknown} \cup \text{Nothing}$) will be used to explicitly denote Nothing is included in a type.

TODO: Consider if the following rule should apply, and similar: $\dfrac{T_1 \in N}{T_1 \cup \text{Unknown} =: \text{Unknown}}$

10.1 Pure Operations on Unknown

Note: Cannot have $y : T \to x + y : T \cup \text{Error}$, 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.

10.1.1.) $$\frac{\vdash x : \text{Unknown} \quad \vdash y : T \quad T \in N}{\vdash x + y : u64 \cup i64 \cup f64 \cup \text{Error}}$$

10.1.2.) $$\frac{\vdash x : \text{Unknown} \quad \vdash y : \text{Unknown}}{\vdash x + y : u64 \cup i64 \cup f64 \cup \text{Error}}$$

Aswell, depending on whether or not Unknown includes Nothing, the following rules might be needed:

10.1.3.) $$\frac{\vdash x : (\text{Unknown} \cup \text{Nothing}) \quad \vdash y : (T \cup \text{Nothing}) \quad T \in N}{\vdash x + y : (u64 \cup i64 \cup f64 \cup \text{Error} \cup \text{Nothing})}$$

10.1.4.) $$\frac{\vdash x : (\text{Unknown} \cup \text{Nothing}) \quad \vdash y : (\text{Unknown} \cup \text{Nothing})}{\vdash x + y : (u64 \cup i64 \cup f64 \cup \text{Error} \cup \text{Nothing})}$$

10.2 Stateful Operations on Unknown

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

10.2.1.) $$\frac{\vdash x : \text{Unknown}}{\vdash \min(x) : u64 \cup i64 \cup f64 \cup \text{Error}}$$

10.2.2.) $$\frac{\vdash x : \text{Unknown}}{\vdash \max(x) : u64 \cup i64 \cup f64 \cup \text{Error}}$$

10.2.3.) $$\frac{\vdash x : \text{Unknown}}{\vdash \text{sum}(x) : u64 \cup i64 \cup f64 \cup \text{Error}}$$

10.2.4.) $$\frac{\vdash x : \text{Unknown}}{\vdash \text{product}(x) : u64 \cup i64 \cup f64 \cup \text{Error}}$$

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

10.3 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:

10.2.1.) $$\frac{\vdash x : \text{Unknown} \quad \vdash k : T}{\vdash x[k] : \text{Unknown} \cup \text{Error}}$$

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.

10.4 Propagation of Error Type

All pure and stateful operations must propagate the Error type. This would apply to all existing rules as a secondary rule, broadly defined as:

10.4.1.) $\dfrac{\vdash a : T_1 \cup \text{Error} \quad \vdash b : T_2}{\vdash f(a,b) : \text{typeof}(f(T_1, T_2)) \cup \text{Error}}$ (and it's commutative corrolary)

10.4.2.) $\dfrac{\vdash a : T \cup \text{Error}}{\vdash f(a) : \text{typeof}(f(T)) \cup \text{Error}}$

For some specific examples:

10.4.3.) $\dfrac{\vdash a : T \cup \text{Error}}{\vdash \text{array}(a) : \{\}\{K_n(u32) : T\} \cup \text{Error}}$

10.3.4.) $\dfrac{\vdash a : T \cup \text{Error}}{\vdash \text{count}(a) : u32 \cup \text{Error}}$