Julia Subtyping: A Rational Reconstruction

FRANCESCO ZAPPA NARDELLI, Inria and Northeastern U.
JULIA BELYAKOVA, Czech Technical U. in Prague
ARTEM PELENITSYN, Czech Technical U. in Prague
BENJAMIN CHUNG, Northeastern U.
JEFF BEZANSON, Julia Computing
JAN VITEK, Northeastern U. and Czech Technical U. in Prague

COMPLETE LIST OF ISSUES REPORTED TO JULIA DEVELOPERS

The complete list of the issues we reported to the Julia bug tracker since starting this project follows. For each report, the number (and active link) in parentheses is the issue id's in Julia's github database. We distinguish between bug reports that have been fixed, bug reports that have been acknowledged and for which a solution is currently being investigated, and other design improvement proposals.

0.1 Fixed Bugs

(1) **Reflexivity and transitivity broken due to buggy diagonal rule** (#24166)
Flaws in the implementation of the diagonal rule check lead invalidate expected properties of the subtype relation, as discussed in Sec. **??**. These flaws are observable in Julia 0.6.2 but have been fixed in the development version.

(2) **Propagation of constraints when subtype unions** (#26654)
The order of types inside a Union constructor should not affect the subtype relation (a property we call symmetry of Union). The subtype algorithm however traverses the types inside a Union constructor in a precise order. Incorrect propagation of constraints during subtyping made subtyping dependent on the order of types inside a Union constructor, as highlighted by the Julia 0.6.2 behavior below:

```
julia> Ref{Union{Int, Ref{Number}}} <: Ref{Union{Ref{T}, T}} where T
true
```

This issue was found by our fuzz tester. It has been fixed in the development version.

(3) **`Union{Ref{T}, Ref{T}}` and `Ref{T}` behave differently** (#26180)
This bug was introduced after the Julia 0.6.2 release:

```
julia> Ref{Union{Ref{Int}, Ref{Number}}} <: Ref{Ref{T}} where T
false

julia> Ref{Union{Ref{Int}, Ref{Number}}} <: Ref{Union{Ref{T}, Ref{T}}} where T
true
```

The second check should return `false`, as the first one, because the two types on the right-hand side are equivalent. This bug was found by our fuzz tester. It has been fixes in the development version (with the same commit that fixes the previous bug report).

0.2 Open Issues

(1) **Missing intersection types** (#26131)

```
julia> Vector{Vector{Number}}  <:
       Vector{Union{Vector{Number}, Vector{S}}} where S<:Integer
true
```

As discussed in Sec. **??**, this query should return `false` because `Vector{S}` is not a subtype of `Vector{Number}` when `Vector{S}<: Integer`. To correctly derive similar judgments, the subtype algorithm must be able to compute the intersection of types. This is a hard problem in itself. As a temporary band-aid, in reply to our call, Julia developers have introduced a `simple_meet` function which computes intersections for simple cases. The current implementation is still too weak to handle this particular case. The fact that not computing the intersection of the upper bounds in rule R_LEFT might be source of problems in presence of union types was suggested by an anonymous reviewer; our example is built on top of reviewer's remark.

(2) **Stack overflows / Loops in subtype.c** `subtype_unionall`
Unexpected looping inside the subtype algorithm, or large computer-generated types, can make Julia subtype algorithm to exceed the space allocated for the recursion stack. We reported this issue on a large computer-generated type (#26065). We discovered later that other reports address a similar issue; some are referenced in the ticket above, but some are more recent (#26487).

(3) **Inconsistent constraints are ignored** (#24179)
Frontend simplification rewrites types of the form `T where lb<:T<:ub` into the upper bound `ub`, without checking first if the user-specified bounds are inconsistent, as in:

```
julia> T where String<:T<:Signed
Signed
```

This may lead to unexpected results in subtype queries, and the type above is not considered equivalent to the `Union{}`. Julia developers agree this behavior is incorrect.

(4) **Diagonality is ignored and constraints are missing when matching with union** (#26716)
Both Julia 0.6.2 and 0.7-dev incorrectly return `true` on these judgments (on the left types are equivalent, on the right it is the same type):

```
julia> (Tuple{Q,Bool} where Q<:Union{Int,P} where P) <: Tuple{Union{T,Int}, T} where T
true

julia> (Tuple{Union{Int,P},Bool} where P) <: Tuple{Union{T,Int}, T} where T
true

julia> (Union{Tuple{Int,Bool}, Tuple{P,Bool}} where P) <: Tuple{Union{T,Int}, T} where T
true
```

The correct answer is `false` because the variable `T` should be considered diagonal and gets matched both with `P` and `Bool`, and as such it cannot be concrete. This is confirmed by rewriting into an equivalent type by the *lift_union* function, thus making the diagonal variable explicit. In this case Julia returns the correct answer.

0.3 Proposals we made

(1) **Interaction of diagonal rule and lower bounds** (#26453)

Whenever the programmer specifies explicitly a lower bound for a type-variable, as in `Tuple{T,T}where T>:t`, it is not always easy to decide if `T` should be considered diagonal or not. This depends on whether the lower bound, `t`, is concrete, but in general deciding concreteness is hard and Julia implementation approximates it with an heuristic. We proposed that the variables should be considered diagonal only if their concreteness is obvious. The proposal was approved, implemented and merged into the `master` branch.

(2) **Another approach to fix problem with concreteness of `Vector{T}` / transitivity** (comment #372746252).
A subtle interaction between the bottom type and the diagonal rule can break transitivity of the subtype relation. We propose an alternative approach to fix the issue, as the solution to the problem applied in Julia seems unsatisfactory.