Sorry that I did not communicate for last month. The reason of that is I did not know what to ask or not know what's missing, along with other things that distracted me. To summarize, I was reading various papers to fill in more knowledge on DOT, and a book [1].

1 Some algebraic analysis

The reason I started to read this book to try to understand how misbehaving DOT can be if a bad-bound occurs. Specifically, I want to answer following question:

Question 1. To which point the type lattice will be jeopardized if a bad bound is accepted?

It turns out that, some simple lattice algebraic analysis shows the following:

Lemma 1. Any bad bound collapses the type lattice into a single point.

This requires that bad bound involves two types whose relation we know do not look like what bad bound says. Let me illustrate what my thought is in greater details. This analysis is by hand so I try to be very careful about it.

First, we can form a complete lattice out of DOT's concrete types:

Definition 1. [1, p. 2.4] For a non-empty ordered set P, for all $S \subseteq P$, there is a supremum($\bigvee S$) and infimum($\bigwedge S$) of S, then P is a complete lattice.

We can define both supremum and infimum very intuitively. Then we can reason about the lattice just by using supremum and infimum operations. The interesting part of this theory is it tells a lot of what an equivalence class should look like without having to specify what defines the equivalence relation, and that the ordered set is. In this context, we only care about one equivalence relation $A =:= B$, which is implied by $A <: B, B <: A$. We can check that it satisfies reflexivity, symmetry and transitivity, so it's indeed an equivalence relation.

Moreover, this equivalence relation is a congruence.

Definition 2. [1, p. 6.5] A congruence is an equivalent relation that's compatible with supremum and infimum.

$A =:= B$ is congruence, because it's effectively defined based on the deductive closure of the consequence where $A =:= B$ holds. Following describes what congruences should look like in lattices.

Lemma 2. [1, p. 6.14] θ is a congruence on lattice L if and only if each block of θ is a sublattice of L.

Following is a specialization and a convenient form of argument that I use a lot when doing reasoning.

Lemma 3. [1, pp. 6.13, 6.14] **The quadrilateral argument:** Consider following diagrams, where $a \to b$ means $a \leq b$ for some order relation. Bold line asserts some equivalence relation($\equiv$).

Figure 1: quadrilateral argument

The argument says that any one of the pairs in the equivalence relation will force the other one in the pair to be also in the equivalence relation.

Proof. It can be shown by straightforward equational reasoning. For the diagram on the left, if $a \equiv a \vee b$, then $a \wedge b \equiv (a \vee b) \wedge b = b$. The second derivation is due to $b \leq a \vee b \Rightarrow a \vee b = b$. Other directions can be proved symmetrically. $\square$

If we treat concrete types in DOT in such form, then we can apply lots of tools to reason about it.

Lemma 4. Bad bound must introduce equivalences.

Proof. There are two types of bad bound, one type is when $A <: B$, the program introduces $B <: A$ somehow, then in this case, we have acquired the new equivalence, so it's trivial in this case; the other type is when A, B are not comparable, then $A <: B$ is asserted. We can see that this also implies equivalence relation.

Figure 2: bad bound between two irrelevant types

The left shows the original relation. Since DOT forms complete lattice, we know $A \vee B$ exists; when we force $A <: B$, we turn left into diagram on the right. Then, it asserts that $A \vee B =:= B$. Since we assumed that A, B are not comparable, we know $A \vee B =:= B$ originally does not hold, hence a new equivalence. $\square$

Therefore, we know that whenever bad bound occurs, we must have introduced new equivalence relation somehow. However, attention needs to be paid here by actually defining what a bad bound is. We call a bound bad, because it "makes no sense". To spell it out, both sides of the bound already have certain relation that we know (either subtype relation or no relation), which contradicts to what the bad bound says. In that sense, it's clearly that we won't call the following program introduces a bad bound:

```
μ x : {
  A : x.B .. x.C
  B : x.C .. x.A
  C : x.A .. x.B
}
```

Because path types are, intuitively, not considered "concrete", therefore it has freedom to be whatever it is. In another word, if a bad bound occurs, we are saying that some two concrete types. Having this in mind, we can see how bad bound behaves by proving the claim (lemma 1) in the beginning.

Proof. We have four kinds of concrete types, $\top$, $\perp$, objects and functions. Any equivalence relation can involve any pair of them. We have 8 non-trivial cases, and following can be easily discharged: $\top =:= \perp$, object =:= functions.

The case for collapsing $\top$ and objects is very general. Consider following quadrilateral argument (F is any function type):

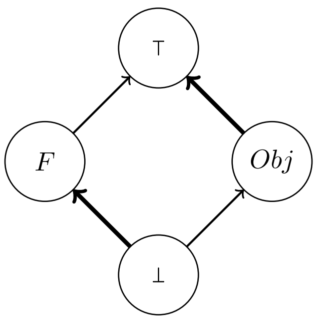

Figure 3: quadrilateral argument for function and object types

This holds because all functions are incomparable with any object. Therefore their supremum and infimum must be $\top$ and $\perp$ respectively. Now applying quadrilateral argument shows F and $\perp$ must collapse. Note that F is not specified, therefore it applies for all function types. By transitivity of equivalence relation, all function types become equivalent. Specifically, $\forall(x : \top)\top$ and $\forall(x : \top)\bot$ collapse, which implies $\top$ and $\bot$ also collapse.

This pattern is very general. Due to the incomparability between function types and object types, we can show that if any one of them collapse with either $\top$ or $\bot$, then we can infer $\top$ and $\bot$ also collapse. This rules out following cases: $\top$ =:= object, $\top$ =:= function, $\bot$ =:= object, $\bot$ =:= function.

We have only two cases left, function collapse with function and object with object. The former introduces two smaller equivalent relation, and therefore can be dischargable by induction.

Then what's left is to show that collapsing two different objects implies the diagram above. But it's not difficult. There are only two relations between two objects: either one is subtype of another, or they are not comparable, which are captured by following quadrilateral arguments:

Figure 4: object =:= object

Consider the first case. If we know $T_2 <: T_1$, then we know for those fields they intersect, they must either be the same, or possess some proper subtyping relation. We can split two cases according to this, either there are at least one fields in both T_1, T_2 such that one in T_2 is a further refinement of one in T_1; or T_2 must have fields that do not exist in T_1, and we can aggregate those fields in T_3. Following are example of each case:

```
T₁ := μ x : {
  A : ⊥ .. ⊤
}
```

```
T₂ := μ x : {
  A : ⊥ .. { μ y : { AA : ⊥ .. ⊤ } }
  B : ⊥ .. ⊤
}
```

We can see that A is refined, so we can only $T_2 <: T_1$; while in the case where such refinement does not occur, it's captured by the left diagram above:

```
T₁ := μ x : {
  A : ⊥ .. ⊤
}
```

```
T₂ := μ x : {
  A : ⊥ .. ⊤
  B : ⊥ .. ⊤
}
```

The latter case has shown that T_3 =:= $\top$ just like the diagram on the left. For the former case, then there must be at least one fields that is refined in T_2 and the reason for the bad bound. This then can be discharged by induction to find out what's the type equivalent to $\top$ or $\bot$.

For the other diagram, if we assert T_1 =:= T_2, then by lemma 2, the smallest sublattice must be in the same equivalence class. Specifically, we can derive $T_1 \wedge T_2$ =:= $T_1 \vee T_2$ and they cannot be the same, assuming T_1 =:= T_2 is introduced by bad bound. If $T_1 \vee T_2$ is $\top$ then we are done; otherwise, we fall back to the previous case, and we've learned that there must be another type that becomes equivalent to one of the extrema.

We've exhausted all the cases at this point, and the original claim is proved. □

lemma 1 is very helpful, it basically says if there is indeed a lattice structure in the type system, then we can conclude the following:

Corollary 1. With bad bound's presence, all terms can be typed to $\bot$.

So this treatment of bad bounds is very tempting. It matches what intuition says: once we detect a bad bound, which is determined to not being able to instantiate, why bother typing the program under its influence in the first place? $\bot$, then, becomes a flag that says "following program is nonsense". From algebra, there seems to be a clearer track on what can be done to make typing easier. So my question is following,

Question 2. Do you know any similar algebraic analysis done on calculi with subtyping?

However, the current DOT does not have this structure. From algebraic perspective, it seems to show the current definition has following problems:

1. In above discussion, I implicitly fit all objects into a sublattice. This does not work in current DOT, due to each recursive type form its own little blob. I was mentioning to change this, which will be discussed below.
2. The intersection types introduces lots of equivalent types that are syntactically not the same. For example, in above argument, If $F \wedge Obj \neq \bot$, then the proof breaks, and it's what happens in DOT. $F \wedge Obj$ has its own right to be a type in DOT, except that it cannot be instantiated, and consequently it becomes a bottom that is less bottom.
3. Intersection type is also not quite consistent. In the case of record types, intersection type functions like a supremum, while not quite the case for any other types. The problem here is the intersection type becomes a part of the syntax, instead of a computation.

To interpret it from another side, we can say that we want to make the types in DOT to form a well studied structure such that certain analysis can be done "for free". I think these suggest that removing intersection type from type primitive and unifying object types are good moves, which are what I am doing.

2 Modifying DOT

Besides removing intersection, I changed DOT's types as following:

$$
\begin{aligned}
T &:= \ldots \ // \ \top, \bot \text{ function and selection remain the same.} \\
&\quad \mid \mu(x : DS) \\
DS &:= [D] \mid D :: DS \\
D &:= \ldots \ // \text{ declarations as before}
\end{aligned}
$$

It effectively turns the declarations into a list that has at least one element. The typing rule for this is routine. The subtyping rules are following:

$$\frac{\Gamma, x : \mu(x : \{a : T\} :: DS) \vdash T <: U}{\Gamma \vdash \mu(x : \{a : T\} :: DS) <: \mu(x : \{a : U\} :: DS)} \ \textbf{(OBJ-FIELD)}$$

$$\frac{\Gamma, x : \mu(x : \{A : S_1..T_1\} :: DS) \vdash S_2 <: S_1 \qquad \Gamma, x : \mu(x : \{A : S_1..T_1\} :: DS) \vdash T_1 <: T_2}{\Gamma \vdash \mu(x : \{A : S_1..T_1\} :: DS) <: \mu(x : \{A : S_2..T_2\} :: DS)} \ \textbf{(OBJ-TYPE)}$$

Both **OBJ-FIELD, OBJ-TYPE** need straightforward variations for $[D]$.

$$\frac{}{\Gamma \vdash \mu(x : DS_1 + +DS_2) <: \mu(x : DS_2)} \ \textbf{(OBJ-DROP1)}$$

$$\frac{}{\Gamma \vdash \mu(x : DS_1 + +DS_2) <: \mu(x : DS_1)} \ \textbf{(OBJ-DROP2)}$$

$$\frac{\Gamma \vdash \mu(x : DS) <: \mu(x : DS_1) \qquad \Gamma \vdash \mu(x : DS) <: \mu(x : DS_2)}{\Gamma \vdash \mu(x : DS) <: \mu(x : DS_1 + +DS_2)} \ \textbf{(OBJ-MERGE)}$$

$$\frac{\Gamma \vdash x : \mu(x : \{A : S..T\} :: DS)}{\Gamma \vdash x.A <: T} \ \textbf{(SEL1)} \qquad \frac{\Gamma \vdash x : \mu(x : \{A : S..T\} :: DS)}{\Gamma \vdash S <: x.A} \ \textbf{(SEL2)}$$

Currently the rules are still not deterministic, specifically transitivity is still around. Hopefully these rules are enough to replace the rules for original intersection types. I originally thought that proving soundness could be easy, by simply translating both terms and types from the modified version to the original one, but it's not true. This modification has changed the type lattice entirely, so the language expressed by this language is not the same one anymore. Therefore proving it sound is basically working out another new calculus. I won't expect it to be at the same level of difficulty, though, since **REC-I, REC-E** are removed completely. So my question on this part is,

Question 3. Should I go ahead and prove this calculus sound?

Notice that, this modification does not necessary jeopardize the expressiveness of the language too badly by taking away intersection type from user level. In fact, one observation is, as long as we can express a type being subtypes of two different types in covariant position, we are automatically talking about intersection type. For example,

```
μ x : {
  A : ⊥ .. ⊤
  B : ⊥ .. ⊤
  C : ⊥ .. A
  D : C .. B
}
```

This has forced $C <: A \wedge D <: A \wedge B$ without spelling it out. It's the same case for union type, whenever we have the ability to talk about relation in contravariant position, we are forced to reason about union type, without having to have language support. Therefore to some extent, this language looks nicer than DOT from different angles, except that some functionality requires more verbose encoding.

However, this language alone seem to lack power to manipulate path types, because we can't explicitly require path type intersecting with some other types. But then this suggests we cannot do it without explicitly defining intersection type and union type, for the sake of dealing with path types, while in the case of no involvement of path types, we eagerly compute the supremum and infimum. Then this suggests another calculus, which possess rules of following kind:

$$\frac{\Gamma \vdash T <: S_1 \qquad \Gamma \vdash T <: S_2}{\Gamma \vdash T <: infimum(S_1, S_2)} \textbf{ (INFIMUM)}$$

In this case, infimum involving path types must use intersection type:

```
infimum x.T x.T = x.T
infimum x.T S = x.T ∧ S
infimum T x.S = T ∧ x.S
...
```

and others similarly. Infimum and supremum computations will be widely spread out in the typing and subtyping rules.

There seems two potential directions to go, hence the question above. The first one is definitely going to be easier to work with, but it's unclear to me where these two paths can lead to.

I will proceed with the first one for now. Let me know if you think it's not a good idea.

3 Conclusion

I think I will keep looking into lattice theory as I found it very fruitful. I still have a number of papers on DOT to read. I wasn't able to get too much from dotty a while ago, but now I developed some theoretic basis, so hopefully for a second time I can get some juice from it.

Please let me know what do you think about it and your suggestions.

References

[1] B. A Davey. *Introduction to lattices and order*. eng. 2nd ed. Cambridge, U.K. ; New York: Cambridge University Press, 2002. ISBN: 0521784514.