Let T be the set of (possible) Python types. Define a relation:

Definition 1. *For $t_1, t_2 \in T$ we say t_1 is* layout compatible *with t_2, written $t_1 \trianglelefteq t_2$ if t_1's description of the memory layout of its instances is safe for use with instances of t_2.*

This is clearly reflexive, and if it's not transitive we're in serious trouble. It's not a partial order as it's not anti-symmetric, but:

Definition 2. *For $t_1, t_2 \in T$ we say t_1 and t_2 are* layout equivalent*, written $t_1 \sim t_2$ if $t_1 \trianglelefteq t_2$ and $t_2 \trianglelefteq t_1$.*

$\trianglelefteq$ defines a partial order on $\sim$-equivalence classes. The `solid_base` function in `Objects/typeobject.c` is a canonical way of choosing a reprentative of a type t's $\sim$-equivalence class, so we can mostly ignore this detail.

Given types t_1 and t_2 the *meet* $t_1 \vartriangle t_2$ is always defined, up to $\sim$, but the *join* $t_1 \triangledown t_2$ may not be – in fact it's only defined when the t_i are $\trianglelefteq$-related in some order and in that case is the $\trianglelefteq$-greater of the t_i.

Definition 3. *A (finite) set $\{t_i\}$ of types is* acceptable for subclassing *if the set has a $\trianglelefteq$-maximal element.*

A subclass of an acceptable $\{t_i\}$ will be $\trianglelefteq$-greater than this maximal element.

Definition 4. *A sequence $t_1, \ldots, t_n$ is an* acceptable MRO for t *if $t_i \trianglelefteq t$ for all $i = 1, \ldots, n$.*

This means, broadly, that if u is an element of an acceptable MRO for t you can use u's description of the memory layout of its instances to describe an instance of t without causing crashes or other erroneous behaviour.

Theorem 1. *If a set of types $\{t_i\}$ is acceptable for subclassing and each t_i has an acceptable MRO, then the default MRO computation for Python will produce an acceptable MRO for the new subclass.*

Proof. Let the MRO of each t_i be written $t_{i1}, t_{i2}, \ldots, t_{in_i}$, so we have $t_{ij} \trianglelefteq t_i$ (each t_i has an acceptable MRO).

Let t be the join of t_i, so $t_i \trianglelefteq t$ ($\{t_i\}$ is acceptable for subclassing).

Let the new subclass of $\{t_i\}$ be u (so $t \trianglelefteq u$).

Let the output of the default MRO computation be $v_1, \ldots, v_k$.

Now the default MRO computation produces a sequence which contains only types already contained in the MRO of one of the t_i or the new subclass itself u.

If $v_l = t_{ij}$ for some i and j, then $v_l = t_{ij} \trianglelefteq t_i \trianglelefteq t \trianglelefteq u$. Trivially, if $v_l = u$, $v_l = u \trianglelefteq u$.

So $v_1, \ldots, v_k$ is an acceptable MRO for u. □

So, does this formalism match up with what the code does? Potential problems:

- is layout compatibility really transitive?
- does the code compute layout compatibility correctly?

I'd be *reasonably* – but not completely – confident of both if we ignore varadic types. I don't know whether it would be better to try to prove the code we have now computes what it thinks it does, or to rewrite it using language closer to what I have used in this note.