Property (4) is a reformulation for *ObjectTypeOf* of property (3) of Section 2.4, the subclassing-is-subtyping property. While property (3) is a double implication, the converse of (4) does not hold: there may be unrelated c and c' such that $ObjectTypeOf(c) = O$ and $ObjectTypeOf(c') = O'$, with $O' <: O$.

Therefore we have partially decoupled subclassing from subtyping by relaxing the double implication (3) to the single implication (4). Subclassing still implies subtyping, so all the previous uses of subsumption are still allowed. But, since subsumption is based on subtyping and not subclassing, we now have even more freedom in subsumption.

In conclusion, the notion of subclassing-is-subtyping can be weakened to subclassing-implies-subtyping without loss of expressiveness, and with a gain in separation between interfaces and implementations.

3.3 Type Parameters

Type parameterization is a general technique for reusing the same piece of code at different types. It is becoming common in modern object-oriented languages, in part independently of object-oriented features. We describe it here both because it is interesting in its own right, and because it is used for the treatment of binary methods later in this chapter.

In conjunction with subtyping, type parameterization can be used to remedy some typing difficulties due to contravariance, for example in method specialization. Consider the object types *Person* and *Vegetarian*, and assume *Vegetables* <: *Food* (but not vice versa) [34]:

> **ObjectType** *Person* **is**
>   …
>   **method** *eat*(*food*: *Food*);
> **end**;
>
> **ObjectType** *Vegetarian* **is**
>   …
>   **method** *eat*(*food*: *Vegetables*);
> **end**;

The intention is that a vegetarian is a person, so we would expect *Vegetarian* <: *Person*. However, this inclusion cannot hold because of the contravariance on the argument of the *eat* method. If we erroneously assume *Vegetarian* <: *Person*, then a vegetarian can be subsumed into *Person*, and can be made to eat meat.

We can obtain some legal subsumptions between vegetarians and persons by converting the corresponding object types into type operators (functions from types to types) parameterized on the type of *food*: