When this same technique is used to compare generic type references, two type references are considered the same when they originate in the same declaration and have identical type arguments.

In certain circumstances, generic types that directly or indirectly reference themselves in a recursive fashion can lead to infinite series of distinct instantiations. For example, in the type

```
interface List<T> {
    data: T;
    next: List<T>;
    owner: List<List<T>>;
}
```

'List<T>' has a member 'owner' of type 'List<List<T>>', which has a member 'owner' of type 'List<List<List<T>>>', which has a member 'owner' of type 'List<List<List<List<T>>>>' and so on, ad infinitum. Since type relationships are determined structurally, possibly exploring the constituent types to their full depth, in order to determine type relationships involving infinitely expanding generic types it is necessary to introduce a mechanism that terminates the recursion.

Within a generic class or interface *G*, type references contained in declared or inherited members are classified as follows

- A type reference without type arguments or with type arguments that do not reference any of *G*'s type parameters is classified as a ***closed*** type reference.
- A type reference that references any of *G*'s type parameters in a type argument is classified as an ***open*** type reference.
- A type reference which, directly or indirectly, references *G* through open type references and which contains a wrapped form of any of *G*'s type parameters in one or more type arguments is classified as an ***infinitely expanding*** type reference. A type is said to wrap a type parameter if it references the type parameter but isn't simply the type parameter itself.

A type is said to originate in an infinitely expanding type reference if it was created (by instantiation of a generic class or interface) from an infinitely expanding type reference (in that generic class or interface).

When comparing two types *S* and *T* for identity (section 3.8.2), subtype (section 3.8.3), and assignability (section 3.8.4) relationships, if either type originates in an infinitely expanding type reference and the other is not the Any, Null, or Undefined type, *S* and *T* are not compared by the rules in the preceding sections. Instead, for the relationship to be considered true,

- *S* and *T* must both be type references to the same named type, and
- the relationship in question must be true for each corresponding pair of type arguments in the type argument lists of *S* and *T*.

Likewise, when making type inferences (section 3.8.6) from a type *S* to a type *T*, if either type originates in an infinitely expanding type reference, then