consist of a set of candidate type arguments collected for each of the type parameters. The inference process recursively relates *S* and *T* to gather as many inferences as possible:

- If *T* is one of the type parameters for which inferences are being made, *S* is added to the set of inferences for that type parameter.
- Otherwise, if *S* and *T* are object types, then for each member *M* in *T*:
    - If *M* is a property and *S* contains a property *N* with the same name as *M*, inferences are made from the type of *N* to the type of *M*.
    - If *M* is a call signature, no other call signatures exist in *T*, exactly one call signature *N* exists in *S*, *N* is non-generic, and the number of required parameters in *N* is greater than or equal to that of *M*, then inferences are made from parameter types in *N* to parameter types in the same position in *M*, and from the return type of *N* to the return type of *M*.
    - If *M* is a construct signature, no other construct signatures exist in *T*, exactly one construct signature *N* exists in *S*, *N* is non-generic, and the number of required parameters in *N* is greater than or equal to that of *M*, then inferences are made from parameter types in *N* to parameter types in the same position in *M*, and from the return type of *N* to the return type of *M*.
    - If *M* is a string index signature and *S* contains a string index signature *N*, inferences are made from the type of *N* to the type of *M*.
    - If *M* is a numeric index signature and *S* contains a numeric index signature *N*, inferences are made from the type of *N* to the type of *M*.

3.8.7 Recursive Types

Classes and interfaces can reference themselves in their internal structure, in effect creating recursive types with infinite nesting. For example, the type

```
interface A { next: A; }
```

contains an infinitely nested sequence of 'next' properties. Types such as this are perfectly valid but require special treatment when determining type relationships. Specifically, when comparing types *S* and *T* for a given relationship (identity, subtype, or assignability), the relationship in question is assumed to be true for every directly or indirectly nested occurrence of the same *S* and the same *T* (where same means originating in the same declaration and, if applicable, having identical type arguments). For example, consider the identity relationship between 'A' above and 'B' below:

```
interface B { next: C; }

interface C { next: D; }

interface D { next: B; }
```

To determine whether 'A' and 'B' are identical, first the 'next' properties of type 'A' and 'C' are compared. That leads to comparing the 'next' properties of type 'A' and 'D', which leads to comparing the 'next' properties of type 'A' and 'B'. Since 'A' and 'B' are already being compared this relationship is by definition true. That in turn causes the other comparisons to be true, and therefore the final result is true.