- *M* is a property and *S'* contains a property *N* where
  - *M* and *N* have the same name,
  - the type of *N* is a subtype of that of *M*,
  - *M* and *N* are both public or both private, and
  - if *M* is a required property, *N* is also a required property.
- *M* is an optional property and *S'* contains no property of the same name as *M*.
- *M* is a non-specialized call or construct signature and *S'* contains a call or construct signature *N* where
  - the signatures are of the same kind (call or construct),
  - the number of non-optional parameters in *N* is less than or equal to that of *M*,
  - *N* can be successfully instantiated in the context of *M* (section 3.8.5),
  - each parameter type in the instantiation of *N* is a subtype or supertype of the corresponding parameter type in *M* for parameter positions that are present in both signatures, and
  - the result type of *M* is Void, or the result type of the instantiation of *N* is a subtype of that of *M*.
- *M* is a string index signature of type *U* and *S'* contains a string index signature of a type that is a subtype of *U*.
- *M* is a numeric index signature of type *U* and *S'* contains a string or numeric index signature of a type that is a subtype of *U*.

When comparing call or construct signatures, parameter names are ignored and rest parameters correspond to an unbounded expansion of optional parameters of the rest parameter element type.

Note that specialized call and construct signatures (section 3.7.2.4) are not significant when determining subtype and supertype relationships.

Also note that type parameters are not considered object types. Thus, the only subtypes of a type parameter *T* are *T* itself and other type parameters that are directly or indirectly constrained to *T*.

3.8.4 Assignment Compatibility

Types are required to be assignment compatible in certain circumstances, such as expression and variable types in assignment statements and argument and parameter types in function calls.

S is ***assignable to*** a type *T*, and *T* is ***assignable from*** *S*, if one of the following is true, where *S'* denotes the apparent type (section 3.8.1) of *S*:

- *S* and *T* are identical types.
- *S* or *T* is the Any type.
- *S* is the Undefined type.
- *S* is the Null type and *T* is not the Undefined type.
- *S* or *T* is an enum type and the other is the primitive type Number.
- *S* and *T* are type parameters, and *S* is directly or indirectly constrained to *T*.
- *S'* and *T* are object types and, for each member *M* in *T*, one of the following is true: