4.12.1 Overload Resolution

The purpose of overload resolution in a function call is to ensure that at least one signature is applicable, to provide contextual types for the arguments, and to determine the result type of the function call, which could differ between the multiple applicable signatures. Overload resolution has no impact on the run-time behavior of a function call. Since JavaScript doesn't support function overloading, all that matters at run-time is the name of the function.

The compile-time processing of a typed function call consists of the following steps:

- First, a list of candidate signatures is constructed from the call signatures in the function type in declaration order. For classes and interfaces, inherited signatures are considered to follow explicitly declared signatures in `extends` clause order.
    - A non-generic signature is a candidate when
        - the function call has no type arguments, and
        - the signature is applicable with respect to the argument list of the function call.
    - A generic signature is a candidate in a function call without type arguments when
        - type inference (section 4.12.2) succeeds in inferring a list of type arguments,
        - the inferred type arguments satisfy their constraints, and
        - once the inferred type arguments are substituted for their associated type parameters, the signature is applicable with respect to the argument list of the function call.
    - A generic signature is a candidate in a function call with type arguments when
        - The signature has the same number of type parameters as were supplied in the type argument list,
        - the type arguments satisfy their constraints, and
        - once the type arguments are substituted for their associated type parameters, the signature is applicable with respect to the argument list of the function call.
- If the list of candidate signatures is empty, the function call is an error.
- Otherwise, if the candidate list contains one or more signatures for which the type of each argument expression is a subtype of each corresponding parameter type, the return type of the first of those signatures becomes the return type of the function call.
- Otherwise, the return type of the first signature in the candidate list becomes the return type of the function call.

A signature is said to be an ***applicable signature*** with respect to an argument list when

- the number of arguments is not less than the number of required parameters,
- the number of arguments is not greater than the number of parameters, and
- for each argument expression *e* and its corresponding parameter *P,* when *e* is contextually typed (section 4.19) by the type of *P*, no errors ensue and the type of *e* is assignable to (section 3.8.4) the type of *P*.