4.12.2 Type Argument Inference

Given a signature $< T_1 , T_2 , \dots , T_n > (p_1 : P_1 , p_2 : P_2 , \dots , p_m : P_m)$, where each parameter type *P* references zero or more of the type parameters *T*, and an argument list $(e_1 , e_2 , \dots , e_m)$, the task of type argument inference is to find a set of type arguments $A_1 \dots A_n$ to substitute for $T_1 \dots T_n$ such that the argument list becomes an applicable signature.

The ***inferred type argument*** for a particular type parameter is the widened form (section 3.9) of the best common type (section 3.10) of a set of candidate types. In order to compute candidate types, the argument list is processed as follows:

- Initially all inferred type arguments are considered ***unfixed*** with an empty set of candidate types.
- Proceeding from left to right, each argument expression *e* is ***inferentially typed*** by its corresponding parameter type *P*, possibly causing some inferred type arguments to become ***fixed***, and candidate type inferences (section 3.8.6) are made for unfixed inferred type arguments from the type computed for *e* to *P*.

The process of inferentially typing an expression *e* by a type *T* is the same as that of contextually typing *e* by *T*, with the following exceptions:

- Where expressions contained within *e* would be contextually typed, they are instead inferentially typed.
- Where a contextual type would be included in a candidate set for a best common type (such as when inferentially typing an object or array literal), an inferential type is not.
- When a function expression is inferentially typed (section 4.9.3) and a type assigned to a parameter in that expression references type parameters for which inferences are being made, the corresponding inferred type arguments to become ***fixed*** and no further candidate inferences are made for them.
- If *e* is an expression of a function type that contains exactly one generic call signature and no other members, and *T* is a function type with exactly one non-generic call signature and no other members, then any inferences made for type parameters referenced by the parameters of *T*'s call signature are ***fixed*** and, if *e*'s call signature can successfully be instantiated in the context of *T*'s call signature (section 3.8.5), *e*'s type is changed to a function type with that instantiated signature.

In the example

```
function choose<T>(x: T, y: T): T {
    return Math.random() < 0.5 ? x : y;
}

var x = choose("Five", 5);
```

inferences for 'T' in the call to 'choose' are made as follows: For the first parameter, an inference is made from type 'string' to 'T'. For the second parameter, an inference is made from type 'number' to 'T'. Since