parameter or return type annotation, member accessor parameter or return type annotation, or index signature type annotation.

- An interface directly depends on each *Type* specified as a type parameter constraint, each *TypeReference* specified as a base interface, and the *ObjectType* specified as its body.
- A module directly depends on its exported members.
- A *Type* or *ObjectType* directly depends on every *TypeReference* that occurs within the type at any level of nesting.
- A *TypeReference* directly depends on the type it references and on each *Type* specified as a type argument.

A named type *T* having a root module *R* (section 2.3) is said to be ***at least as accessible as*** a member *M* if

- *R* is the global module or an external module, or
- *R* is an internal module in the parent module chain of *M*.

In the example

```
interface A { x: string; }

module M {
    export interface B { x: A; }
    export interface C { x: B; }
    export function foo(c: C) { … }
}
```

the 'foo' function depends upon the named types 'A', 'B', and 'C'. In order to export 'foo' it is necessary to also export 'B' and 'C' as they otherwise would not be at least as accessible as 'foo'. The 'A' interface is already at least as accessible as 'foo' because it is declared in a parent module of foo's module.

10.5 Declaration Merging

Internal modules are "open-ended" and internal module declarations with the same qualified name relative to a common root (as defined in section 2.3) contribute to a single module. For example, the following two declarations of a module outer might be located in separate source files.

File a.ts:

```
module outer {
    var local = 1;           // Non-exported local variable
    export var a = local;    // outer.a
    export module inner {
        export var x = 10;   // outer.inner.x
    }
}
```

File b.ts: