```
var v1: G<A, C>;                // Ok
var v2: G<{ a: string }, C>;    // Ok, equivalent to G<A, C>
var v3: G<A, A>;                // Error, A not valid argument for U
var v4: G<G<A, B>, C>;          // Ok
var v5: G<any, any>;            // Ok
var v6: G<any>;                 // Error, wrong number of arguments
var v7: G;                      // Error, no arguments
```

A type argument is simply a *Type* and may itself be a type reference to a generic type, as demonstrated by 'v4' in the example above.

As described in section 3.5, a type reference to a generic type *G* designates a type wherein all occurrences of *G*'s type parameters have been replaced with the actual type arguments supplied in the type reference. For example, the declaration of 'v1' above is equivalent to:

```
var v1: {
    x: { a: string; }
    y: { a: string; b: string; c: string };
};
```

3.6.3 Type Queries

A type query obtains the type of an expression.

TypeQuery:
    `typeof` *TypeQueryExpression*

TypeQueryExpression:
    Identifier
    TypeQueryExpression `.` *IdentifierName*

A type query consists of the keyword `typeof` followed by an expression. The expression is restricted to a single identifier or a sequence of identifiers separated by periods. The expression is processed as an identifier expression (section 4.3) or property access expression (section 4.10), the widened type (section 3.9) of which becomes the result. Similar to other static typing constructs, type queries are erased from the generated JavaScript code and add no run-time overhead.

Type queries are useful for capturing anonymous types that are generated by various constructs such as object literals, function declarations, and module declarations. For example:

```
var a = { x: 10, y: 20 };
var b: typeof a;
```

Above, 'b' is given the same type as 'a', namely '{ x: number; y: number; }'.

3.6.4 Type Literals

Type literals compose other types into new anonymous types.