3.11 Implicit Parameters

3.11.1 Basic Principle

- Many operators (e. g., `min` and `max`) cannot be completely generic, because their implementation needs particular operators (e. g., comparison operators) which are not available for all types.
- Such operators can be passed as *implicit parameters*, however, for example:

```
[(T:type)]                              $$ Deducible parameter
min (x:T) (y:T)
[(+ (T) "<=" (T) -> (bool))]            $$ Implicit parameter
-> (T = if x <= y then x else y end)
```

Explanations

- In analogy to § 3.8.1, `(+ (T) "<=" (T) -> (bool))` is a declaration of an anonymous parameter (which in turn has two anonymous parameters itself) of the operator `min`.
- In contrast to normal operator declarations, where a leading plus sign or backslash is meaningless, a plus sign at the begin of a parameter declaration marks the parameter as an implicit parameter, which is sensibly also optional.

3.11.2 Assignment of Implicit Parameters

- In an application of the `min` operator such as `min 1 2`, a *matching* operator from the current context is passed automatically, i. e., an operator which can exactly replace the parameter (cf. § 3.9.1).
- Because the type `int` has already been deduced for the parameter `T` according to the operands `1` and `2`, an operator must be passed here which can replace an operator of type `(int) "<=" (int) -> (bool)`, which is possible, for example, for the chained comparison operator for `int` values defined on a task sheet.
- Applications of the implicit parameter in the implementation of the operator (the expression `x <= y` in the example) are then forwarded to the passed operator.
- If there is no matching operator in the current context, the expression is erroneous. For example, `min 'a' 'z'` would be erroneous, because there is no operator that can replace an operator of type `(char) "<=" (char) -> (bool)`.
- This error could be remedied, however, by defining such an operator in advance, for example:

```
(x:char) "<=" (y:char) -> (bool = int x <= int y)
```

- By that means, implicit parameters can be used to express additional conditions or constraints for the types of other parameters of an operator, and, if necessary, an operator can also have multiple implicit parameters.

- If there is more than one matching operator for an implicit parameter in the current context, the most specific one (cf. § 3.9.2) is passed if it exists; otherwise, the expression would be ambiguous in that case.

Notes

- If a parameter declaration does not start with a plus sign, it does not declare an implicit parameter, and, therefore, no matching operator is passed automatically (and, accordingly, it is not an error if there is no matching operator in that case).
- If the parameter is nevertheless optional and no matching operator is passed explicitly (cf. § 5.1.1), the parameter is assigned, according to § 3.4, the value nil, i. e., a nil operator that does not have an implementation and that returns a new synthetic value for every application (cf. § 4.2).
- Initially, there are no exclusions for the operators defined by implicit parameters except for the exclusion of the predefined sequential evaluation mentioned in § 3.6. It is possible, however, to define appropriate exclusions inside the implementation of the operator to which these parameters belong.

3.11.3 Indirect Implicit Parameters

- If an implicitly passed operator has implicit parameters itself, matching operators from the current context must also be passed for them to make the respective expression correct (and so on, if these operators have implicit parameters in turn, and so forth).

Example: Double Square Operator

```
[(T:type)] (x:T) "²" [(+ (T) "*" (T) -> (T))] -> (T = x * x);
[(T:type)] (x:T) "⁴" [(+ (T) "²" -> (T))] -> (T = (x²)²)
```

- For the expression `10`^{4}, the square operator $\bullet^2$ is implicitly passed to the double square operator $\bullet^4$, and the predefined operator for multiplication, division, and remainder of `int` values (cf. § 3.5.3) is in turn implicitly passed to the square operator (where this operator is more general than the implicit parameter $\bullet * \bullet$).

- For the expression `'x'`^{4}, a multiplication operator for `char` values would have to be passed accordingly, which is usually not available, causing the expression to be erroneous.

Example: Output of Variables

```
[(T:type)]
print <o>[only] (x:T?)
[(+ print only (T) -> (bool))]
-> (bool =
  print only ?x;
  <o>[true | print 1:0]
)
```

- To be able to print the value of a variable `x` of type `T?`, the above operator `print` needs a corresponding output operator for the content type `T` that is passed as an implicit parameter.
- Because `1:0` is equal to nil, `print 1:0` simply prints a line terminator (cf. § 2.12).
- Exemplary use:

```
i : int?; i =! 1; print i;                $$ 1
ii : int??; ii =! i; print ii;            $$ 1
iii : int???; iii =! ii; print iii        $$ 1
```

- For the expression `print i`, the content type `T` is deduced as `int` and, therefore, the predefined operator `print` is passed to the implicit parameter (where this operator is more general than the parameter).

- For the expression `print ii`, the content type `T` is deduced as `int?` and, therefore, the operator `print` for variables is passed to its own implicit parameter.
  For this operator, `T` is then deduced as `int` and, therefore, the predefined operator `print` is passed to its implicit parameter.

- For the expression `print iii`, the content type `T` is deduced as `int??` and, therefore, the operator `print` for variables is again passed to its own implicit parameter.
  For this operator, `T` is then deduced as `int?` and, therefore, the operator `print` for variables is again passed to its implicit parameter.
  For this operator, `T` is finally deduced as `int` and, therefore, the predefined operator `print` is passed to its implicit parameter.

- In the same way, variables with any number of “levels” can be printed in principle, if there is an output operator for their “final“ content type.

- However, the compiler aborts the recursive search for assignments of implicit parameters at a certain depth to avoid endless recursions for “malicious” programs.