Porting the Oberon Compiler from Oberon to Oberon-07

Niklaus Wirth, 15. 8. 2007

After defining the Revised Oberon called Oberon-07 and implementing it, I have expressed the compiler in the revised language. This should seem a trivial endeavour, but it is worth noting a few experiences.

Features that had been removed

1. The Loop Statement

There were only 4 loop statements in the compiler – a sign of good programming ☺. The principal characteristic of the loop statement is that it allows several termination points. Thus, there is no general recipe for translating it into a repeat or while statement. In my experience, a careful search for an new formulation is worth while. In all cases, the new solution turned out to be more satisfactory.

2. Forward declarations of procedures

First, one should see if the forward reference can be eliminated by a mere exchange of procedure declarations, or by a nesting. If, for example, P calls Q and Q calls P, the forward reference might possibly be eliminated by nesting, namely if one of the procedures is called only from within the other:

```
PROCEDURE P;

    PROCEDURE Q;
    BEGIN …. P …….
    END Q;

BEGIN …. Q ….
END P
```

A general solution is the introduction of a procedure variable. If, for example, P(…) would require a forward declaration, declare

```
VAR p: PROCEDURE (…);
```

and (in the same scope) initialize it with the assignment p:= P, and replace all calls of P by calls of p.

3. SHORTINT, LONGINT, and LONGREAL

Change all occurrences of SHORTINT and LONGINT to INTEGER. This may require the elimination of calls of type transfers SHORT and LONG, an exercise in text editing. LONGREAL has not been implemented in Oberon-SA. Replace it by REAL, but do not expect exactly the same results ☺.

4. Structured value parameters

Structured value parameters imply the copying of the parameter's value and therefore should be avoided if possible. But strings are structured constants and therefore cannot be passed as VAR parameters – although some implementations allow it – because the string's value might be changed by assignment to the formal parameter. To remedy this defect, Oberon now features a third kind of parameter, the Const parameter. Hence, if strings are to be passed to a formal parameter of type ARRAY OF CHAR, the symbol CONST must be used as a prefix.

5. Standard procedures INC, DEC, INCL, EXCL, COPY, ASH, SHORT, LONG

INC(x, y)	→	x := x + y, unless y is a constant (0 <= y < 256)
DEC(x, y)	→	x := x – y
INCL(s, k)	→	s := s + {k}
EXCL(s, k)	→	s := s – {k}
COPY(x, y)	→	y := x
ASH(x, n)	→	LSL(x, n), if n > 0
	→	ASR(x, n), if n < 0

Features that had been restricted or changed

1. The case statement

The new case statement requires that

1. the case expression and the labels be of type INTEGER
2. the labels form a contiguous range, starting with 0
3. the number of cases must be specified in the case clause

The compiler contained only 4 case statements. The transformation of labels of type CHAR to type INTEGER in the scanner was tedious, but a one-time exercise in editing. The explicit specification of the number of cases, which yields the jump table array size, is trivial.

2. Export of variables

The export of variables is stylistically a dubious practice, because it contradicts the principle of "information hiding" and of "abstract data types". External access to variables should occur through exported procedures only. In order to read a value, however, direct export of the variable does not infringe on information hiding, because it does not permit the violation of module invariants, while making access faster. Hence, Oberon-07 allows variable export in read-only mode.

In order to alter the value of an imported variable, a corresponding procedure must be introduced and exported.

New features

1. Set complement and inclusion

The set of set operators has been completed with set complement and set inclusion. -*s* denotes the complement of *s*; *s0* <= *s1* (or s1 >= s0) denotes inclusion.

2. The extended While statement

The while statement has been extended according to the suggestion of the late Edsger Dijkstra. Note that it terminates, when all conditions are FALSE. This facility may. for example, be used to express the following loop construct:

```
LOOP
    IF b0 THEN S0
    ELSIF b1 THEN S1
    ELSIF b2 THEN S2
    ELSE EXIT
    END
END
```

in a simpler way, without nesting:

```
WHILE b0 DO S0
ELSIF b1 DO S1
ELSIF b2 DO S2
END
```

3. The standard procedures PACK, UNPK, and ASSERT

PACK(x, e) is a new standard procedure to change the exponent of a floating-point number x. Its effect is equal to x := x * 2^e, and it is achieved by simply adding *e* to the exponent of *x*.

UNPK(x, e) does the inverse of PACK. It assigns to *e* the exponent of *x*, and divides *x* by 2^e, such that 1.0 <= x < 2.0.

ASSERT(b, n) traps, if the Boolean expression *b* yields FALSE. The integer *n* serves the operating system to output this value, such that the programmer can easily determine which assertion failed. *n* can be omitted.