Calculation patterns for the design of loops

Tom Verhoeff*

February 2003, August 2004, Translated from Dutch in April 2016

1 Introduction

In this note, I discuss a couple of calculation patterns[1] that appear in the course *Design of Algorithms 1* [3] when designing[2] loops. We investigate the shape of loop invariants, the accompanying calculations, and the way these are related.

Consider the following program fragment[3], being a repetition with 'hole' $\mathcal{E}$.

```
{ inv: P }
do B
→ { P ∧ B }
    v := E
;   { P(n := n + 1) }
    n := n + 1
    { P }
od
{ P ∧ ¬B }
```

Here, $\mathcal{E}$ is a *program* expression to be determined, while invariant P and guard B have already been chosen. Initially, we will focus on finding $\mathcal{E}$ driven by the need to guarantee invariance of P, and we will not worry about finding P and B, initialization, finalization, and termination. For the latter, see §6.

As a running example, we use the following specification for exponentiation[4].

```
|[  con X, N: int;
    var v: int;
▷   { pre: 0 ≤ N }
    Power
    { post: v = X^N }
]|
```

Given integer constants X and N, with $0 \leq N$, the program *Power* needs to assign X^N to integer variable v.

*Software Engineering & Technology, Dept. of Math. & CS, Eindhoven University of Technology, T.Verhoeff@tue.nl

[1] This term is inspired by the term *design pattern* [1]. Meyer states in [2, p. 72] that a successful design pattern should be a directly (re)usable software component and not only a description on paper. That is why we speak of calculation patterns, and not design patterns, in this note.

[2] The course concerns the —calculational— construction of program fragments, and not *a posteriori* verification of correctness.

[3] We follow the notation in [4].

[4] Exponentiation is not an available program operator; only addition and multiplication are allowed.

2 The basic pattern

The approach to find $\mathcal{E}$ is always the same: Start proving the correctness of the program and while doing so find a suitable $\mathcal{E}$ that can serve to complete the proof.

The first pattern, which I call the **basic pattern**, can always be applied, regardless of the shape of the invariant P. We determine $\mathcal{E}$ by plainly applying the *Axiom of Assignment* to $v := \mathcal{E}$ with the relevant pre- and postcondition. The proof obligation is then[5]:

$$[\ P \wedge B \quad \Rightarrow \quad P(n := n+1)(v := \mathcal{E})\ ]$$

Here, $Q(v := \mathcal{E})$ denotes the predicate Q in which every free occurrence of v has been replaced by $\mathcal{E}$.

If no further knowledge about the shape of P is available, then it is best[6] to organize the proof as follows.

$$\begin{array}{ll} & |[\ P \wedge B \\ & \triangleright \\ & \quad P(n := n+1)(v := \mathcal{E}) \\ = & \quad \{\ \text{definition of } P\text{, substitution}\ \} \\ & \quad \ldots \\ = & \quad \{\ \bullet\ \text{Choose } \mathcal{E} = \ldots\ \} \\ & \quad \ldots \\ = & \quad \{\ \ldots\ \} \\ & \quad \text{true} \\ &]| \end{array}$$

The predicate before $\triangleright$ is called the *context condition*, which is assumed to hold in the subsequent calculation. The calculation is written in the style of Edsger W. Dijkstra, Wim Feijen, and Netty van Gasteren.

Calculations of the following shape are usually *not recommendable*, because one cannot freely make use of the precondition $P \wedge B$ in all calculation steps.

$$\begin{array}{ll} & \quad P(n := n+1)(v := \mathcal{E}) \\ = & \quad \{\ \text{definition of } P\text{, substitution}\ \} \\ & \quad \ldots \\ \Leftarrow & \quad \{\ \bullet\ \text{Choose } \mathcal{E} = \ldots\ \} \\ & \quad \ldots \\ \Leftarrow & \quad \{\ \ldots\ \} \\ & \quad P \wedge B \end{array}$$

And the following approach is (almost) *never convenient*, because the formulae become so wide and unwieldy, and most parts (especially $P \wedge B \Rightarrow \ldots$) will hardly change.

$$\begin{array}{ll} & \quad P \wedge B \ \Rightarrow\ P(n := n+1)(v := \mathcal{E}) \\ = & \quad \{\ \text{definition of } P\text{, substitution}\ \} \\ & \quad \ldots \\ = & \quad \{\ \bullet\ \text{Choose } \mathcal{E} = \ldots\ \} \\ & \quad \ldots \\ = & \quad \{\ \ldots\ \} \\ & \quad \text{true} \end{array}$$

[5] For convenience' sake, we assume that definedness of $\mathcal{E}$ does not play a role.

[6] The context assumption $P \wedge B$ can sometimes be weakened.

3 The conjunction pattern

If we do know something about the shape of P, then it may be possible to exploit this in the presentation of the calculations.

> We speak of the **conjunction pattern** when P has the shape of a conjunction, say $P0 \wedge P1$.

Applying the basic pattern to this leads to the following calculation.

$$
\begin{array}{ll}
& |[\ P0 \wedge P1 \wedge B \\
& \triangleright \\
& \quad (P0 \wedge P1)(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ substitution distributes over } \wedge \ \} \\
& \quad P0(n := n+1)(v := \mathcal{E}) \ \wedge \ P1(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ definition of } P0 \text{ and } P1, \text{ substitution } \} \\
& \quad \ldots \\
= & \quad \{ \bullet \text{ Choose } \mathcal{E} = \ldots \} \\
& \quad \ldots \\
= & \quad \{ \ldots \} \\
& \quad \text{true} \\
&]|
\end{array}
$$

To limit the width of the calculation, we rather split it into two separate calculations for $i \in \{ 0, 1 \}$ of the shape:

$$
\begin{array}{ll}
& |[\ P0 \wedge P1 \wedge B \\
& \triangleright \\
& \quad Pi(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ definition of } Pi, \text{ substitution } \} \\
& \quad \ldots \\
= & \quad \{ \bullet \text{ Choose } \mathcal{E} = \ldots \} \\
& \quad \ldots \\
= & \quad \{ \ldots \} \\
& \quad \text{true} \\
&]|
\end{array}
$$

Of course, both choices of $\mathcal{E}$ must be compatible. Often, v does not appear in both $P0$ and $P1$. In that case there can be no dilemma for $\mathcal{E}$, because $\mathcal{E}$ appears only if v occurs.

We have taken the complete invariant and guard as context assumption in both (sub)calculations. Often, a weaker assumption can suffice in each subcalculation (also see the next example). If in the calculation for Pi one only needs to assume $Pi \wedge B$, then Pi is an invariant *in isolation*. If $P0$ and $P1$ both are invariants in isolation, then $P0 \wedge P1$ is also an invariant. But conversely, if $P0 \wedge P1$ is an invariant, then neither $P0$ nor $P1$ needs to be an invariant in isolation.

Subsequently, whenever we speak of an invariant P, we often do not mean to imply that P is an invariant *in isolation*, but rather that it is a *conjunct of* an invariant. Possibly, *other* conjuncts are needed to establish the invariance of P. Sometimes such other conjuncts are already known in advance (think of bounds on variables); in other cases these will only be discovered during the subsequent calculations (think of strengthening of invariants [4, §4.3]).

Observe that $a \leq b \leq c$ is equivalent to $a \leq b \ \wedge \ b \leq c$ and that hence the conjunction pattern applies as well.

For example, for exponentiation we can have:

$$\begin{array}{lcl} B & : & n \neq N \\ P & : & 0 \leq n \leq N \end{array}$$

From splitting P into $0 \leq n \wedge n \leq N$, and applying the conjunction pattern, it appears that $0 \leq n$ and $n \leq N$ are invariants in isolation:

$$\begin{array}{l} |[\ 0 \leq n \\ \triangleright \\ \quad (0 \leq n)(n := n+1)(v := \mathcal{E}) \\ = \quad \{\ \text{substitution}\ \} \\ \quad 0 \leq n+1 \\ = \quad \{\ 0 \leq n \text{ from context}\ \} \\ \quad \text{true} \\ ]| \end{array} \qquad \begin{array}{l} |[\ n \leq N \ \wedge \ n \neq N \\ \triangleright \\ \quad (n \leq N)(n := n+1)(v := \mathcal{E}) \\ = \quad \{\ \text{substitution}\ \} \\ \quad n+1 \leq N \\ = \quad \{\ n \leq N \text{ and } n \neq N \text{ from context}\ \} \\ \quad \text{true} \\ ]| \end{array}$$

4 The head pattern

We speak of the **head pattern**[7] if (the relevant conjunct in) the invariant has the following shape:

$$v \quad = \quad F$$

where v is a program variable and F an expression in terms of the *other* program variables, i.e. F does not depend on v.

In terms of our program fragment, we prefer to write

$$v \quad = \quad \varphi.n$$

where φ is an appropriately chosen function. In the course [3] (but not in [4]) this is called the φ-scheme[8]. In the example of exponentiation we can have:

$$\varphi.n \quad = \quad X^n$$

If you apply the basic pattern, then you obtain a calculation of the following shape.

$$\begin{array}{l} |[\ P \wedge B \\ \triangleright \\ \quad P(n := n+1)(v := \mathcal{E}) \\ = \quad \{\ \text{definition of } P\ \} \\ \quad (v = \varphi.n)(n := n+1)(v := \mathcal{E}) \\ = \quad \{\ \text{substitution, } v \text{ does not depend on } n, \text{ and } \varphi.n \text{ not on } v\ \} \\ \quad \mathcal{E} = \varphi.(n+1) \\ = \quad \vdots \\ \quad \mathcal{E} = \ldots \\ = \quad \{\ \bullet \text{ Choose } \mathcal{E} = \ldots\ \} \\ \quad \text{true} \\ ]| \end{array}$$

[7] Counterpart of the tail pattern; also see §7.

[8] This name was coined by Wim Feijen.

The choice for $\mathcal{E}$ then becomes straightforward. For example, for exponentiation with $P : v = X^n$ we calculate as follows:

$$
\begin{array}{ll}
& |[\ P \wedge B \\
& \triangleright \\
& \quad P(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ definition of } P \ \} \\
& \quad (v = X^n)(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ substitution } \} \\
& \quad \mathcal{E} = X^{n+1} \\
= & \quad \{ \text{ property of exponentiation } \} \\
& \quad \mathcal{E} = X^n * X \\
= & \quad \{ \ v = X^n \text{ from context (see } P) \ \} \\
& \quad \mathcal{E} = v * X \\
= & \quad \{ \bullet \text{ Choose } \mathcal{E} = v * X \ \} \\
& \quad \text{true} \\
&]|
\end{array}
$$

As such, this calculation is correct, but the presentation can be improved.

What is striking in the shape of this calculation is that the left-hand side of the formulae, i.e., $\mathcal{E} = \ldots$, does not change. You can take that part 'outside' the calculation. In [4], this happens for the first time in the calculation at the top of page 58, where this 'taking outside' happens without further explanation, and only in the conclusion is that simplified calculation related to the invariance proof.

The *universal part* of the calculation in the head pattern, which need not be made explicit every time, consists of:

$$
\begin{array}{ll}
& |[\ P \wedge B \\
& \triangleright \\
& \quad P(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ definition of } P \ \} \\
& \quad (v = \varphi.n)(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ substituion, } v \text{ does not depend on } n, \text{ and } \varphi.n \text{ not on } v \ \} \\
& \quad \mathcal{E} = \varphi.(n+1) \\
= & \quad \{ \bullet \text{ Choose } \mathcal{E} = \varphi.(n+1) \ \} \\
& \quad \text{true} \\
&]|
\end{array}
$$

Note that the choice $\mathcal{E} = \varphi.(n+1)$ does not mean that we literally choose $\varphi.(n+1)$ for $\mathcal{E}$. We are also allowed to choose something that is equivalent to it. The purpose of rewriting $\varphi.(n+1)$ is to arrive at a *program* expression. That rewriting is always relevant to show.

The calculation for the head pattern $v = \varphi.n$ is presented as follows.

$$
\begin{array}{ll}
& |[\ P \wedge B \\
& \triangleright \\
& \quad \varphi.(n+1) \\
= & \quad \{ \ldots \} \\
& \quad \vdots \\
= & \quad \{ \ldots \} \\
& \quad \ldots \\
&]|
\end{array}
$$

The calculation must result in an appropriate program expression, which we can then substitute for $\mathcal{E}$.

The calculation for the head pattern applied to exponentiation with $\varphi.n = X^n$ then boils down to this:

$$
\begin{array}{ll}
& |[\ P \wedge B \\
& \triangleright \\
& \quad \varphi.(n+1) \\
= & \quad \{ \text{ definition of } \varphi \ \} \\
& \quad X^{n+1} \\
= & \quad \{ \text{ property of exponentiation } \} \\
& \quad X^n * X \\
= & \quad \{ \ v = X^n \text{ from context (see } P) \ \} \\
& \quad v * X \\
&]|
\end{array}
$$

From the context of this calculation, one can immediately see under which precondition the result is valid. This is of importance for the order in which the statements of the derived program can be written. Omitting the context, or only partially completing the calculation increases the mental burden (both of the designer and later readers). Although the following calculation does capture the main argument in a recurrence equation for φ, the transformation to the complete program text is more involved than with the preceding calculation. We can only recommend this to experienced designers.

$$
\begin{array}{ll}
& \varphi.(n+1) \\
= & \quad \{ \text{ definition of } \varphi \ \} \\
& \quad X^{n+1} \\
= & \quad \{ \text{ property of exponentiation } \} \\
& \quad X^n * X \\
= & \quad \{ \text{ definition of } \varphi \ \} \\
& \quad \varphi.n * X
\end{array}
$$

5 The tail pattern

We speak of the **tail pattern**[9] when (the relevant conjunct in) the invariant has the following shape:

$$F \quad = \quad C$$

where F is an expression in terms of the program variables and C is a constant, i.e. C does not depend on program variables that change in the loop.

In terms of our program fragment we prefer to write it as

$$F.v.n \quad = \quad C$$

where F is an appropriately chosen *function* of the program variables. In the example of exponentiation, we can[10] take:

$$\begin{array}{rcl} F.v.n & = & v * X^{N-n} \\ C & = & X^N \end{array}$$

If you apply the basic pattern to such an invariant, then you obtain a calculation of the following shape.

$$\begin{array}{ll} & |[\ P \wedge B \\ & \triangleright \\ & \quad P(n := n+1)(v := \mathcal{E}) \\ = & \quad \{\ \text{definition of } P\ \} \\ & \quad (F.v.n = C)(n := n+1)(v := \mathcal{E}) \\ = & \quad \{\ \text{substitution, } C \text{ does not depend on } v \text{ and } n\ \} \\ & \quad F.\mathcal{E}.(n+1) = C \\ = & \quad \{\ F.v.n = C \text{ from context (see } P)\ \} \\ & \quad F.\mathcal{E}.(n+1) = F.v.n \\ = & \quad \vdots \quad \text{calculation involving } F.v.n \\ & \quad F.\mathcal{E}.(n+1) = F.(\ldots).(n+1) \\ = & \quad \{\ \bullet \text{ Choose } \mathcal{E} = \ldots\ \} \\ & \quad \text{true} \\ &]| \end{array}$$

The tail pattern is less intuitive than the head pattern, because there is more freedom to calculate, which in turn makes it easier to get lost. We have deliberately chosen to eliminate C as soon as possible, and only then to start calculating with $F.v.n$. If the involved operations have *inverses* or certain *cancelation properties* hold, then alternatives are possible. But in general, it is better not to let these alternatives distract you.

[9] The name is inspired by the occurrence of this pattern with tail invariants; also see §7.

[10] Invariant $F.v.n = v * X^n$ with $n := n - 1$ in the body leads to another (more elegant) program.

For example for exponentiation with $P : v * X^{N-n} = X^N$ we calculate[11]:

$$
\begin{array}{ll}
|[& P \wedge n \neq N \quad \wedge n \leq N \\
\triangleright & \\
& P(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ definition of } P \ \} \\
& \left(v * X^{N-n} = X^N\right)(n := n+1)(v := \mathcal{E}) \\
= & \quad \{ \text{ substitution } \} \\
& \mathcal{E} * X^{N-n-1} = X^N \\
= & \quad \{ \ v * X^{N-n} = X^N \text{ from context (see } P) \ \} \\
& \mathcal{E} * X^{N-n-1} = v * X^{N-n} \\
= & \quad \{ \text{ property of exponentiation, } N-n-1 \geq 0 \text{ follows from context } \} \\
& \mathcal{E} * X^{N-n-1} = v * \left(X * X^{N-n-1}\right) \\
= & \quad \{ \text{ associativity of } * \ \} \\
& \mathcal{E} * X^{N-n-1} = (v * X) * X^{N-n-1} \\
= & \quad \{ \bullet \text{ Choose } \mathcal{E} = v * X \ \} \\
& \text{true} \\
]| &
\end{array}
$$

As such, this calculation[12] is correct, but the presentation can be improved.

What is striking in the shape of this calculation is that the left-hand side of the formulae starting with $F.\mathcal{E}.(n+1) = \ldots$ does not change. That part can be taken 'outside'. In [4] this observation is made in a special case of the tail pattern on pages 75–76.

The universal part of the calculation with the tail pattern is harder to describe in isolation than with the head pattern. Therefore, we will not even make an attempt. That universal part need not be presented every time, but the rewriting of the right-hand side $F.v.n$ into $F.(\ldots).(n+1)$ must be presented.

The calculation for the tail pattern $F.v.n = C$ is presented as follows.

$$
\begin{array}{ll}
|[& P \wedge B \\
\triangleright & \\
& F.v.n \\
= & \quad \{ \ldots \} \\
& \quad \vdots \\
= & \quad \{ \ldots \} \\
& F.(\ldots).(n+1) \\
]| &
\end{array}
$$

The calculation should result in a shape of F with as right-hand argument $n+1$ and as left-hand argument an appropriate program expression, which we can then substitute for $\mathcal{E}$ in the program.

[11] To avoid division by zero, the context assumption has been strengthened with $n \leq N$. This must follow from the precondition of $v := \mathcal{E}$, which is possible by also strengthening the invariant with $n \leq N$. Condition $X \neq 0$ would also suffice. But that is not the topic of this exposition.

[12] Observe that we have found the same program expression as via the head pattern.

This is the way to apply the tail pattern to exponentiation with $F.v.n = v * X^{N-n}$:

$$
\begin{array}{l}
|[\ n \neq N \quad \wedge n \leq N \\
\triangleright \\
\quad F.v.n \\
= \quad \{ \text{ definition of } F \ \} \\
\quad v * X^{N-n} \\
= \quad \{ \text{ property of exponentiation, } N - n > 0 \text{ because } n < N \text{ from context } \} \\
\quad v * \left(X * X^{N-n-1}\right) \\
= \quad \{ \text{ associativity of } * \ \} \\
\quad (v * X) * X^{N-n-1} \\
= \quad \{ \text{ definition of } F \ \} \\
\quad F.(v * X).(n+1) \\
]|
\end{array}
$$

Observe that we have not made use of P from the context. This is generally the case with the tail pattern. The invariant $P : F.v.n = C$ is used in the universal part to eliminate C. The guard $B : n \neq N$ and other parts of the total invariant (that in one shape or another may end up in the precondition of the assignment $v := \mathcal{E}$) possibly do play a role (here e.g. $n \leq N$ instead of $N - n > 0$).

6 Initialization, finalization, termination

We have identified these calculation patterns when dealing with invariance. Let us now address initialization, finalization, and termination.

Consider invariant $v = \varphi.n$, which occurs with the head pattern. For postcondition $v = \varphi.N$, finalization can be achieved directly with guard $B : n \neq N$, since

$$[\ v = \varphi.n \ \wedge \ n = N \quad \Rightarrow \quad v = \varphi.N\ ]$$

Initialization of the invariant $v = \varphi.n$ requires calculation of $\varphi.A$, where A must be an appropriate starting value for n. For example, for exponentiation with $\varphi.n = X^n$ and initialization of n with $n := 0$, we calculate:

$$
\begin{array}{l}
|[\ 0 \leq N \\
\triangleright \\
\quad \varphi.0 \\
= \quad \{ \text{ definition of } \varphi \ \} \\
\quad X^0 \\
= \quad \{ \text{ property of exponentiation } \} \\
\quad 1 \\
]|
\end{array}
$$

We see the same pattern in this calculation as with invariance according to the head pattern. This shape of calculation is in general applicable in the situation

$$\{\ Q\ \}\ v := \mathcal{E}\ \ ;n := A\ \{\ v = \varphi.n\ \}$$

This is the case both with initialization (A is an appropriate "small" starting value for n) and with invariance (A is $n + 1$, and Q is $v = \varphi.n \ \wedge \ B$). The shape of the calculation is inspired by the shape of the corresponding postcondition $v = \varphi.n$; the shape of the precondition is not of importance here.

Now consider the invariant $F.v.n = C$, where the tail pattern applies. If we have $C = F.V.N$, then initialization of this invariant can be accomplished directly with $v, n := V, N$, since

$$[\,\text{true} \quad \Rightarrow \quad (F.v.n = F.V.N)(v, n := V, N)\,]$$

For example, for exponentiation with $F.v.n = v * X^n$ and postcondition $v = X^N$, an appropriate invariant is $F.v.n = F.1.N$ (indeed, when the loop ends with $n = 0$, we have $v = F.v.n = F.1.N = X^N$; also see further down, with finalization). The initialization can then be accomplished by $v, n := 1, N$.

The tail pattern offers more freedom than the head pattern. Indeed, with the head pattern $v = \varphi.n$, the value of v is uniquely determined by the value of n, while the tail pattern $F.v.n = C$ allows more[13] combinations of v and n. The tail pattern concerns a (general) relation, whereas the head pattern concerns a limited relation, viz. a function.

That extra freedom gives the tail pattern three advantages over the head pattern:

1. Bigger steps, and more kinds of steps are possible;
2. The statement that decreases the variant function need not be precisely determined in advance;
3. Straightforward early termination.

Consider exponentiation with postcondition $v = X^N$ and tail invariant $v * x^n = X^N$ (note the extra program variable x). Initialization can be done with $v, x, n := 1, X, N$. Let us now calculate the guard for finalization:

$$\begin{array}{ll} & v * x^n = v \\ = & \quad \{ \text{ case distinction on whether } v = 0, \text{ property of } * \,\} \\ & v = 0 \;\vee\; x^n = 1 \\ = & \quad \{ \text{ property of exponentiation } \} \\ & v = 0 \;\vee\; x = 1 \;\vee\; n = 0 \end{array}$$

Thus, the strongest possible guard is

$$v \neq 0 \;\wedge\; x \neq 1 \;\wedge\; n \neq 0$$

For the body there are two simple ways to manipulate $v * x^n$ while preserving its shape:

$$\begin{array}{lcll} v * x^n & = & (v * x) * x^{n-1} & \text{if } 0 < n \\ v * x^n & = & v * (x * x)^{n \mathbf{div} 2} & \text{if } n \text{ even} \end{array}$$

Both ways decrease n provided initially $0 < n$ holds; the second way is faster, but less universally applicable. Given $n \neq 0$ in the loop guard, the body can be completed with the selection:

$$\begin{array}{l} \textbf{if } \text{true} \qquad\quad \rightarrow v, n := v * x, n - 1 \\ [\!]\; n \textbf{ mod } 2 = 0 \rightarrow x, n := x * x, n \textbf{ div } 2 \\ \textbf{fi} \end{array}$$

The first guard of the selection can be strengthened to $n \textbf{ mod } 2 \neq 0$ so as to enforce faster termination.

[13] Whether more combinations actually appear during execution is another matter.

7 Conclusion

The choice of names 'head pattern' versus 'tail pattern' suggests a duality, but that is only partly the case. With the head pattern $v = \varphi.n$, you begin the design of the body by calculating $\varphi.(n+1)$ with as goal a program expression, for instance in terms of $\varphi.n$. With the tail pattern $F.v.n = C$ you begin calculating $F.v.n$ (that is, without substitution) and the goal is obtaining the shape $F.(\ldots).(n+1)$ with a program expression on the dots (preferably even with something faster than $n+1$ as the second argument).

I hope this overview will help eliminate calculations without *heads or tails* (:-).

References

[1] E. Gamma, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[2] B. Meyer. *Object-Oriented Software Construction. Second Edition*. Prentice Hall, 1997.

[3] 2IA10, *Design of Algorithms 1*. TU/e, Computer Science (2003–2007). URL: `www.win.tue.nl/~wstomv/edu/2ia10/`.

[4] A. Kaldewaij. *Programming: The Derivation of Algorithms*. Prentice Hall, 1990.