

For the record: Batcher's Baffler

In this note we consider a special sorting routine for array $f(i: 0 \leq i < N)$. The predicate OK is given by

$$OK.i.j \equiv f.i \leq f.j$$

and in the following quantifications over it are implicitly constrained by $0 \leq i < j < N$. The specification for Batcher's Baffler - the algorithm has been invented by K.E. Batcher and has been given this name by David Gries, presumably because he was baffled by it - is

[[$N: \text{int } \{N \geq 0\}$

; [[$f(i: 0 \leq i < N)$ array of int { bag.f = B }

; Batcher's Baffler

{ bag.f = B \wedge ($\underline{A}i :: OK.i.(i+1)$) }

]]

]]

In the following we shall no longer mention the invariance of bag.f = B ; it is trivially maintained as the algorithm only interferes with the array f with the operation

$$\text{Ord.i.j} = \begin{array}{l} \text{if } OK.i.j \rightarrow \text{skip} \\ \text{if } \neg OK.i.j \rightarrow f:\text{swap}(i,j) \\ \underline{f} \{OK.i.j\} \end{array}$$

There are all sorts of ways of expressing our postcondition that f is ascending, but this one is a nice starting point for the invariant

$P_0: (\underline{A}i :: OK.i.(i+t))$

which suggests for Batchier's Baffler the form

"establish $t \geq N$ " $\{P_0\}$
 ; do $t \neq 1 \rightarrow$ "decrease t under invariance of P_0 od.

Let the decrease of t under invariance of P_0 involve the transition from $t=t'$ to $t=t''$ with $t'' < t'$. It would be nice if we could exploit the precondition $(\underline{A}i :: OK.i.(i+t'))$ by keeping it invariant; we can only hope to do so provided it is implied by the postcondition $(\underline{A}i :: OK.i.(i+t''))$. On account of the transitivity of \leq , this implication holds if t'' is a factor of t' . Under that constraint the most modest decrease of t - i.e. the one that strengthens P_0 as little as possible - is halving t ; thus we suggest for the invariant P_1 , given by

$P_1: P_0 \wedge t$ is a power of 2

and for Batchier's Baffler the form

$t := 1$; do $t < N \rightarrow t := 2 \cdot t$ od $\{P_1\}$

; do $t \neq 1 \rightarrow t := t/2$ $\{(\underline{A}i :: OK.i.(i+2 \cdot t))\}$

"restore P_0 " $\{(\underline{A}i :: OK.i.(i+t))\}$

od

The rest of this note is concerned with the

algorithm for "restore P_0 ", constrained by

$$\{P_2: (\underline{A}_i :: \text{OK}.i.(i+2.t))\}$$

restore P_0

$$\{P_0: (\underline{A}_i :: \text{OK}.i.(i+t))\} \quad ;$$

it treats t as a constant (for which its being a power of 2 is no longer significant).

$$\text{With } e.i \equiv (i \bmod 2.t) < t$$

$$\text{we observe } e.i \equiv \neg e.(i+t)$$

and rewrite P_0 as $P_3 \wedge P_4$ with

$$P_3: (\underline{A}_i : e.i : \text{OK}.i.(i+t)) \quad \text{and}$$

$$P_4: (\underline{A}_i : \neg e.i : \text{OK}.i.(i+t)) \quad \text{or, equivalently}$$

$$P_4: (\underline{A}_i : e.i : \text{OK}.(i+t).(i+2.t))$$

The above dichotomy of the desired OK-relations is of interest because the argument pairs $i, (i+t)$ in P_3 - and also those in P_4 - are disjoint on account of $e.i \equiv \neg e.(i+t)$. Hence P_3 (or P_4) can be established by a whole bunch of Ord-operations that could be performed concurrently. (It is the potential for concurrency that makes Batcher's Baffler of interest.) Using \parallel to denote the potentially concurrent combination, we have

$$(0) \quad \{\text{true}\} (\parallel i : e.i : \text{Ord}.i.(i+t)) \{P_3\}$$

and similarly

$$(1) \quad \{\text{true}\} (\parallel i : e.i : \text{Ord}.(i+t).(i+2.t)) \{P_4\}$$

But one cannot achieve $P_3 \wedge P_4$ by executing the above two consecutively: the second operation

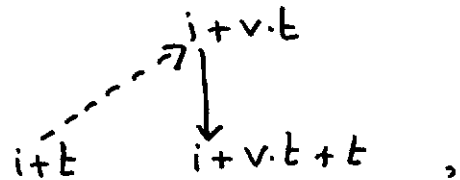
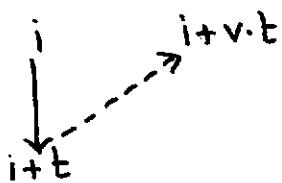
would in general destroy what the first one has accomplished. Let us therefore investigate a strengthening of its precondition such that (1) does not falsify P_3 . For our analysis we generalize (1) by replacing the constant 2 by the parameter v , restricted to even values so as to ensure that all pairs $i, (i+v \cdot t)$ with $e.i$ are disjoint. That is, we consider the operation

$$(2) \quad (\|i: e.i: \text{Ord.}(i+t).(i+v \cdot t))$$

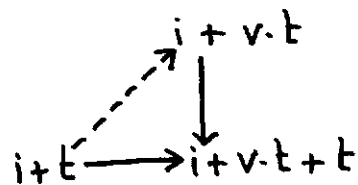
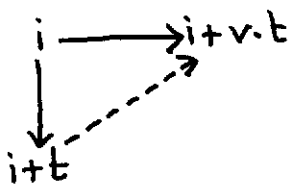
and would like it not to falsify the OK-relations that constitute P_3 . In our analysis we shall use the lemmata and -grudgingly- the notation of EWD 932.

(i) OK-relations with no argument involved in an Ord-operation are maintained on account of EWD 932, Lemma 0.

(ii) Of OK-relations with one argument involved in an Ord-operation we have two types:



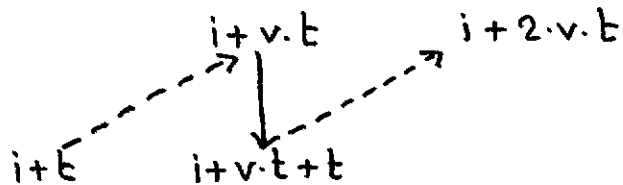
neither of which, however, is a lemma. However



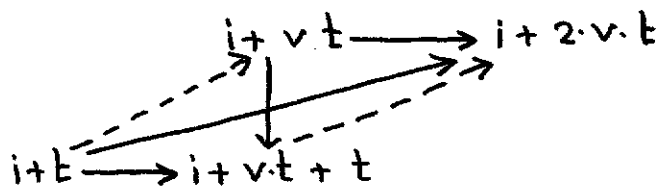
represent EWD 932, Lemma 3. Note that, v being even, the two OK-relations added are implied by P_2 . (Here we are getting our first glimpse of how

to exploit precondition P_2 .)

(iii) Finally we investigate the OK-relation with both arguments involved in an Ord-operation:



This, again, is not a lemma, but we can recognize the sequence $\dashrightarrow \rightarrow \dashrightarrow$ in EWD932, Lemma 4



Two of the added OK-relations are again implied by P_2 , the third one

$$(\underline{A}i: e.i: \text{OK}. (i+t). (i+2.v.t))$$

has to be implied by the precondition of (2), which, besides then leaving P_3 invariant, establishes

$$(\underline{A}i: e.i: \text{OK}. (i+t). (i+v.t))$$

i.e. the same formula after $v := v/2$. We have found the invariant

$$P_5: (\underline{A}i: e.i: \text{OK}. (i+t). (i+v.t)) \wedge v \geq 2 \wedge v \text{ is a power of } 2$$

for "restore P_0 "; it can be initialized by seeing to it that $v.t \geq N$.

We are left with the duty to demonstrate that (0) and (2) maintain P_2 .

For invariance under (0) we have OK-relations

(i) with 0 arguments involved

$$i \longrightarrow i+2 \cdot t \quad \text{EWD932, Lemma 0}$$

(ii) with 1 argument involved

$$\begin{array}{ccc} i & \longrightarrow & i+2 \cdot t \\ \downarrow & & \\ i+t & & \end{array} \quad \text{EWD932, Lemma 1}$$

(iii) with 2 arguments involved

$$\begin{array}{ccc} i & \longrightarrow & i+2 \cdot t \\ \downarrow & & \downarrow \\ i+t & \longrightarrow & i+3 \cdot t \end{array} \quad \text{EWD932, Lemma 2}$$

For invariance under (2) we have OK-relations

(i) with 0 arguments involved

$$i \longrightarrow i+2 \cdot t \quad \text{EWD932, Lemma 0}$$

$$i+t \longrightarrow i+3 \cdot t \quad \text{"}$$

(ii) with 1 argument involved

$$i+v \cdot t - 2 \cdot t \longrightarrow i+v \cdot t \quad \text{EWD932, Lemma 1}$$

$$i+t \dashrightarrow i+v \cdot t \quad \text{"}$$

$$\begin{array}{ccc} i+t & \dashrightarrow & i+v \cdot t \\ & & \downarrow \\ i+t & \longrightarrow & i+3 \cdot t \end{array} \quad \text{"}$$

(iii) with 2 arguments involved

$$\begin{array}{ccc} i+t & \dashrightarrow & i+v \cdot t \longrightarrow i+v \cdot 2 + 2 \cdot t \\ & \longrightarrow & \downarrow \\ i+t & \longrightarrow & i+3 \cdot t \end{array} \quad \text{EWD932, Lemma 2}$$

And now we are ready for the final version of
Batcher's Baffler

```

[[ t, v0: int
; t := 1 ; do t < N → t := 2 · t od ; v0 := 1 {P1 ∧ t · v0 ≥ N}
; do t ≠ 1 → t, v0 := t/2, 2 * v0 {P2 ∧ t · v0 ≥ N}
; (|| i: e.i: Ord. i. (i+t)) {P2 ∧ P3 ∧ t · v0 ≥ N}
; [|| v: int ; v := v0 {P2 ∧ P3 ∧ P5}
; do v ≠ 2 → v := v/2
; (|| i: e.i: Ord. (i+t). (i+v · t))

od
]]

od
]]

```

Remark For $N=4$, Batcher's Baffler generates
the operations described on EWD932-2. (End of
Remark.)

I am indebted to David Gries and the members
of the ETAC (= Eindhoven Tuesday Afternoon Club).

Austin, 9 September 1985

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas, 78712-1188
United States of America