# Johnson's procedure: mechanization and parallelization

M Filali, Nabil Zaidi

# Johnson's procedure: mechanization and parallelization

Mamoun Filali Amine
IRIT
Université de Toulouse, France
Email: filali@irit.fr

Nabil Zaidi
Lycée Blaise Pascal
Clermont-Ferrand, France
Email: nabilzaidi@live.fr

**Abstract**

In this note, we present a mechanization of Johnson's procedure. We first specify the scheduling problem, we are concerned with. Then, we formalize Johnson's procedure and prove an invariant. Thanks to this invariant, we prove the correctness of Johnson's procedure and its optimality. Last, also thanks to the invariant, we elaborate a concurrent version of Johnson's procedure which is correct by construction. We use the Python language for expression and experimentation purposes. The Isabelle HOL assistant theorem prover is used for validating the correctness.

## I. Introduction

Johnson's procedure [1] is one of the oldest procedure that has been proposed for scheduling in an optimal way a list of jobs over two machines. Although Johnson's procedure is described very precisely in the seminal paper [1] and in many scheduling textbooks [2], to the best of our knowledge, the assertional reasoning establishing its correctness with respect to a given program text has not been published. Moreover, we are aware neither of a mechanization[1] of such a proof nor of a parallelization of such a procedure.

This kind of scheduling is used inside multicores for the instructions to be processed. The calculus of the worst case execution time (WCET) for such processors relies on techniques: ILP, constraints resolution, ... for dealing with such a complex problem. Recently, the analysis of the scheduling of trains over rails has also been considered through similar techniques. Also, one technique to address the precise analysis of real time problems is that of timed automata. In order to scale such a technique, it seems promising to reuse the known optimality results for such a problem. Last, if we consider the certification of critical real-time systems, the use of formal methods is by now mandatory (DO178C). The mechanization of the basic procedures of the domain and their parallelization can make easier such certifications. From our point of view, the certification of real-time algorithms and their parallelization is an interesting open problem to address.

## II. Specification

**Notations**

- A list $l$ is a permutation of $l'$ is denoted: $l \sim l'$ ($l$ and $l'$ are also said to be similar). Given two lists $l1$, $l2$, their concatenation is denoted $l1 + l2$. Given an element e, the list consisting of the unique element e ist denoted [e].
- Given a couple $c$ of elements, the first (resp. second) component of $c$ is denoted $c_1$(resp. $c_2$).
- Given a list $l$ of couples, $l$ is non decreasing(resp. non increasing) according to the first element(resp. to the second element) is denoted $\mathbf{inc}_1(l) = \mathbf{inc}(\mathbf{map\ fst}\ l))$ (resp. $\mathbf{dec}_2(l) = \mathbf{dec}(\mathbf{map\ snd}\ l)$).

### A. Expression.

In the following, we model a job by a couple of naturals. The first (resp. the second) element of the couple is the duration required on the first (resp. the second) machine. The jobs to be scheduled are modeled as a list of natural couples.

*1) Makespan definition:* The makespan function $\mathbf{ms}$ is defined through an intermediate definition $\mathbf{ms}_i$ as follows:

$$\begin{array}{ll} \mathbf{ms}_i \in & (\mathbb{N}\times\mathbb{N})\text{list} \times (\mathbb{N}\times\mathbb{N}) \to \mathbb{N}\times\mathbb{N} \\ & (l,i) \mapsto \mathbf{reduce}(\mathbf{lambda}(m,m') : \mathbf{lambda}(d,d') : (m+d, \max(m', m+d) + d')), l, i) \end{array}$$

and $\mathbf{ms}(l) = \mathbf{ms}_i(l, (0,0))$.

*2) Scheduling specification:* We model Johnson's procedure as a function $\mathbf{J}$ with the following properties:

(J1) $\mathbf{J}$ is a total function: $\mathbf{J} \in (\mathbb{N}\times\mathbb{N})\text{list} \to (\mathbb{N}\times\mathbb{N})\text{list}$.
(J2) $\mathbf{J}$ is conservative: $\mathbf{J}$ does not create or loose jobs. $\mathbf{J}$ generates a permutation: $\forall l.\ \mathbf{J}(l) \simeq l$.
(J3) $\mathbf{J}$ is optimal: $\forall l\ l'.\ l \sim l' \Rightarrow \mathbf{ms}(\mathbf{J}(l))_1 \leq \mathbf{ms}(l')_1 \wedge \mathbf{ms}(\mathbf{J}(l))_2 \leq \mathbf{ms}(l')_2$.

[1] Actually, we have done a mechanization of all the development in Isabelle HOL. Due to the lack of space, we do not detail it here.

## III. JOHNSON'S PROCEDURE EXPRESSION AND CORRECTNESS

In order to establish the correctness of Johnson's procedure, we first express it is as a recursive Python procedure. Then, we state an invariant characterizing a schedule as returned by Johnson's procedure.

### A. Johnson's procedure functional expression

```
# auxiliary procedure
def J_aux(b,l,e): # b is to be scheduled first, l remains to process, e is to be scheduled last
    # recursive procedure : the variant is len(l)
    if l == []: return b+e
    else:
        m1,m2 = min (l, key = lambda (d1,d2) : d1 ), min (l, key = lambda (d1,d2) : d2 )
        if m1[0] <= m2[1]: l.remove(m1); return J_aux(b+[m1],l,e)
        else: l.remove(m2); return J_aux(b,l,[m2]+e)
# Johnson's procedure
def J(l): return J_aux([],l,[])
```

### B. Johnson's procedure invariant

The invariant $(I\_J)$ states formally how the schedule returned by Johnson's procedure is structured. This schedule is indeed a 2-partition of the jobs characterized by the comparison ($\leq$) of the first duration and the second duration of a job. The first subset is sorted according to the increasing order of the first duration, while the second subset is sorted according to the decreasing order of the second duration. More formally:

$$
\begin{array}{rl}
 & \mathbf{J}(l) \sim l \\
\wedge & \mathbf{let}\ l1 = [(d1,d2)\ \mathbf{for}\ (d1,d2)\ \mathbf{in}\ \mathbf{J}(l)\ \mathbf{if}\ d1 \leq d2]\ \mathbf{in} \\
(I\_J) & \mathbf{let}\ l2 = [(d1,d2)\ \mathbf{for}\ (d1,d2)\ \mathbf{in}\ \mathbf{J}(l)\ \mathbf{if}\ \ \neg(d1 \leq d2)]\ \mathbf{in} \\
 & \quad \mathbf{J}(l) = l1@l2 \\
 & \wedge\ \ \mathbf{inc}_1(l1) \wedge \mathbf{dec}_2(l2)
\end{array}
$$

### C. Johnson's procedure proof obligations

- The properties (J1) and (J2) are trivially established thanks to the invariant of the auxiliary procedure `J_aux` and the invariant $(I\_J)$ seen in the previous section.
- The proof of property (J3) relies on the following intermediate result stating that the Johnson's procedure is efficient:

  $$\forall l.\ \mathbf{ms}(\mathbf{J}(l))_1 \leq \mathbf{ms}(l)_1 \wedge \mathbf{ms}(\mathbf{J}(l))_2 \leq \mathbf{ms}(l)_2$$

  In turn, this result relies on the following property (and a similar symmetrical one) of the makespan function **ms** wich states when we can swap the jobs of a list while improving the makespan components:

  $$(\forall x\ \mathbf{in}\ l.\ x_1 \geq e_1) \wedge (\forall\ x\ \mathbf{in}\ l.\ x_2 \geq e_1) \wedge e_1 \leq e_2 \Rightarrow \mathbf{ms}([e]+p)_2 \leq \mathbf{ms}(p+[e])_2$$

  Then, (J3) is again obtained thanks to the invariant.

## IV. A CONCURRENT JOHNSON'S PROCEDURE

Thanks to the invariant, we know that the schedule returned by Johnson's procedure is a 2-partition of the jobs. Then, based on this separation property, we propose the following parallelization:

$$\text{partition}\ ;\ (\text{sort}_1 \parallel \text{sort}_2)\ ;\ \text{fusion}$$

where, we first partition the list of jobs, then *concurrently* we sort each partition according to the first and second component. Eventually, we fusion the sorted results. Since this solution validates the invariant and the correctness proofs rely on the invariant, we assert that this solution is correct by construction[2].

## V. CONCLUSION

We have proposed a formal development of Johnson's procedure based on assertional reasoning. The invariant has been used for correctness purposes but also as a hint for parallelizing the procedure. Then, the correctness of the parallel procedure can be obtained for free. We believe that such an approach could be interesting for suggesting the parallelisations of other scheduling algorithms together with their correctness. For future work, we envision the formal validation of other scheduling algorithms, we are especially interested in realtime ones.

## REFERENCES


[1] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, 1954. [Online]. Available: http://dx.doi.org/10.1002/nav.3800010110

[2] P. Brucker, *Scheduling Algorithms*, 3rd ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.


[2]More precisely, we have not looked, a posteriori, for its invariant.