Timetabling for Schools: An ~~Excercise~~ in Program and Data Structuring

C.A.R. Hoare and H. C. Johnston.

Summary. The paper illustrates some recent theories of program and data structuring, with the aid of a realistic large example problem.

0. Introduction.

A number of recent papers [1,2,3] have ~~[illegible]~~ described a method of programming which emphasizes the clear formulation of the objectives of a program and each part of it, a clear justification of design decisions, an orderly ~~app~~ transition from objective and decision to the construction of code. This systematic methodology ~~procedure~~ of program construction is intended to ~~minimise the frequency of programming error, and to produce p~~ produce programs of high quality, with reduced risk of error, good documentation, and at low cost.

~~Most previous examples have concealed~~

Most previous papers have (been) illustrated by ~~their points on~~ a small example program, producing as a result ~~a "logical poem" of~~ what may be called a "logical poem".

~~But~~

In practice, of course, it will be in the construction of large programs that a systematic methodology should pay the highest dividends. ~~This paper makes an a~~ This paper makes a first attempt in this direction progressing ~~progressing~~ ~~[illegible]~~ (it is hoped) from the construction of logical epigrams. to the construction of a logical epic.

The example chosen is one which has some independent interest — that of the construction of timetables for schools. ~~This is intended~~ ~~to be~~ This example exhibits quite a number of the properties which make computer programming difficult but interesting:

(1) Good design of input data and careful validation of it is of prime importance.

(2) ~~Careful analysis of a large~~ The problem requires careful analysis and treatment of a large number of special cases.

(3) The size of the problem, its data, and amount of computation ~~are~~ will stretch the power and capacity of the computers which will be available ~~required~~ to solve it...

~~(4) The A successful~~

(4) At the heart of the problem there is a small but important ~~eleme~~ place for elegant combinatorial programming

(5) A successful technique for solving the problem will not be found by a priori reasoning ~~will~~ but only be ~~found~~ as a result of practical trials of the working programme. ~~Adaptability is therefore of prime importance~~

(6) There is some doubt whether the problem is soluble at all.

Thus the example has been deliberately chosen for its difficulty rather than its simplicity; and should ~~never~~ reveal ~~any~~ clearly the characteristic strengths and weaknesses of the proposed programming methodology.

~~4)~~ The program described in the paper is certainly not ~~a final solution~~ intended to be a final solution, ~~but rather a first attempt~~ to the problem of school timetabling. However, it is hoped that the ~~program~~ documentation which evolves during the construction of the program will be sufficiently and clear and comprehensive to enable other programmers to use, ~~and~~ develop, and improve the ~~methods~~ program described;

~~which are here~~
thus a solution of the problem may evolve, ~~by~~
~~successive improvements made by programmers~~
in the same way as discoveries in other branches
of mathematics, ~~and~~ science, and engineering by
the cooperative endeavours of many scientists in
many places building on the work of their
predecessors. Unless programmers can learn to
stand on ~~each others' shoulders (rather~~
the shoulders of their colleagues (rather than treading
on their toes), the development of ~~Computer~~ difficult computer
~~Science~~ applications is likely to be very slow. ~~[illegible]~~
But ~~But success in this~~

Thus progress will depend on the
willingness of programmers to write programs
of high quality; to explicitly justify the
decisions they take, and to explain their
programs ~~in sou~~ in published articles. ~~This~~
~~article is~~ It is hoped that this article
may serve as a guide and model for
such publications.

(DRAFT)

Timetabling for Schools:
an Exercise in Program and Data Structuring

C.A.R. Hoare and H. C. Johnston

Summary.

The purpose of this paper is to illustrate some recent theories of program and data structuring, with the aid of a realistic large example problem - that of constructing an acceptable timetable for a school.

1. Specification of the Problem.

A school may be described in terms of the items, which constitute the school - teachers, classes, rooms, and equipment. The set of items may be defined for a particular school by enumeration:

```
type Item = {Jones,Smith,...,IV,VA,VB,...,physlab,gym,...,projector,..} ;
```

The number of items will be of order 250.

Some of the items will have more than one unit; for example, there may be two physics laboratories or four projectors which can be used simultaneously. The number units of an item defines the number of possible simultaneous users of an item, and will be given by a mapping:

```
lives: Item → 1..maxlives
```

where maxlives will be typically 8. Most items will have only one life.

The school engages in a number of activities, which are to appear in the timetable;. ~~for example "form VA Latin", or "form IV physics"~~. These activities may be defined by enumeration; // there will usually be less than 500 ~~of them~~. Activities.

newline // type Activity = {form VA Latin, form IV physics, ...} newline

The school timetable is constructed over a week, consisting of a number of periods, usually between 30 and 48, defined by enumeration, for example:

```
type Period = {M1,M2,M3,M4,M5,Tu1,Tu2,...,F7}
```

Each activity will in general be required to occur several times during the week; this is defined by a mapping:

```
times: Activity → 1..maxtimes;
```

where maxtimes will ~~usually be not more than 10~~. be less than size (Period)

Each activity involves participation of a set of items; for example, "IV form physics" will require a meeting of form IV with a teacher, (say Jones); it also requires a physlab, and perhaps also a projector. The requirement for each activity is given by a mapping:

```
requirement: Activity → Item set;
```

We also shall use the inverse of this mapping:

$$\text{users: Item} \rightarrow \text{Activity } \underline{\text{set}}$$

where $a \in \text{users}(i) \equiv i \in \text{requirement}(a)$

A timetable may be specified by giving for each activity the set of periods in which that activity is to take place, that is, by the mapping:

$$\text{timetable: Activity} \rightarrow \text{Period } \underline{\text{set}}$$

This mapping must satisfy the following constraints:

(1) Each activity must take place exactly the right number of times:

$$\forall a\text{:Activity} \quad \text{size}(\text{timetable}(a)) = \text{times}(a) \qquad \text{(C1)}$$

(2) Each Item must be used exactly the right number of times in each period: busy(i,p) = lives(i), for all i,p,

where $\text{busy}(i,p) = \text{size}\{a \mid a \in \text{users}(i) \;\&\; p \in \text{timetable}(a)\}$ (C2)

The requirement that an item may not be underused is not in practice restrictive. If an item i is intended to have free periods, their number can be computed by the formula:

$$\text{size(Period)} \times \text{lives}(i) - \sum_{a \in \text{users}(i)} \text{times}(a)$$

Then for each unit of an item a further artificial activity can be inserted to represent its free periods. A "free" activity is thus an activity which has only one item in its requirement set.

The number of such artificial "free" activities may be up to 250, bringing the total number of activities up to 750.

2. Additional Constraints.

In practice, this relatively simple characterisation of the timetabling problem is complicated by a number of additional constraints, described below:

2.1 Spread.

In all schools, there is a strong disinclination to ~~sch~~ engage in the same activity more than once a day; and ~~therefore~~ an ~~Sa~~ acceptable school timetable must in general ~~always~~ ~~attempts~~ ~~to~~ spread each activity fairly evenly over the week.

~~It will be easier to formulate these constraints if we~~ assume that at most one each multiple period [illegible]

counttuples: 1..maxlength x Period set $\rightarrow$ 0..maxtimes

which counts the number of times a group of the right number of consecutive periods occurs in the period set. The required constraint may now be expressed:

$$\forall a \quad \text{counttuples}(\text{length}(a), \text{timetable}(a)) \times \text{length}(a) = \text{times}(a) \qquad \text{(C5)}$$

~~2.1.~~ Spread.

~~In all school time~~

~~Another~~ constraint is imposed by the desire not to have two ~~occurrences of the same activity on the same day.~~ Activities to which this constraint is applied belong to the set:

spread: Activity set.

~~This~~ constraint, of course, cannot be satisfied by an activity which occurs too often; but we may artificially split such an activity into two or more activities with identical requirements. The days in the week may be defined by enumeration, for example:

type Day = {Monday, Tuesday, Wednesday, Thursday, Friday}

We are also given mappings:

periods in: Day $\rightarrow$ Period set

day of: Period $\rightarrow$ Day

which give the set of periods in a given day, and the day in which any given period occurs. Obviously

$$p \in \text{periods in}(d) \equiv d = \text{day of}(p) \qquad \text{(P1)}$$

$$a \in \text{spread} \supset \text{times}(a) \leq \text{size}(\text{Day}) \; \cancel{\times \text{length}(a)} \qquad \text{(P2)}$$

~~The constraint may now be expressed:~~

$$\forall a \quad a \in \text{spread} \supset \text{size}(\text{possdays}(\text{timetable}(a))) \cancel{\times \text{length}(a)} = \text{times}(a) \qquad \text{(C3)} \; \cancel{\text{(C6)}}$$

where $\text{possdays}(ps) = \{d \mid ps \wedge \text{periods in}(d) \neq \text{empty}\}$

~~Most~~ nonfree activities will be in spread.

The requirement to spread activities may now be expressed:

$$\forall a, d: \quad a \in spread \supset size(timetable(a) \wedge periodsin(d)) \leq 1 \qquad (C3)$$

An alternative (equivalent) formulation of the same constraint is:

$$\forall a: \quad a \in spread \supset size(possdays(timetable(a))) = times(a) \qquad (C4)$$

where $possdays(ps) = \{d \mid ps \wedge periodsin(d) \neq empty\}$

Most non-free activities will be members of the set spread

2.2. Multiperiod activities.

Certain activities of a school ~~may~~ will be such that they can only effectively be carried out in a number of consecutive periods, for example, practical classes, or games. Such constraints may be specified by a mapping:

length:Activity → 1..maxlength~~, where maxlength,~~ (New line) which gives for each activity the length of the multiple period which must be assigned to it; maxlength will usually be 3 or 4.

We also need a mapping:

starts:2..maxlength → Period <u>set</u>

which gives for each length the set of periods in which a multiple period of that length may start (for example, not the periods at the end of a day, or before a lunchbreak). ~~We can now define a function:~~ *

```
*       counttuples(ℓ,ps)function
    begin  new count:+0;
        for  p ∈ starts(ℓ) do {t:=tuple(ℓ,p);
                           if t C ps then   ps:-t;count:+1}
   end  counttuples:=count }
```

where tuple(ℓ,p) is a set of ℓ consecutive periods starting with p.

(~~Now it is not in general~~ we may reasonably ~~assume~~ that acceptable for two multiperiod activities to occur on the same day. ?

such that timetable_0(a) consists of those periods in which ~~that~~ activity a must occur in the completed timetable, i.e.,

$$\forall a \qquad \overset{\text{preassigned}}{\cancel{\text{timetable}}}_0(a) \subset \text{timetable}(a) \qquad \cancel{(C3)}\ (C7)$$

timetable_0

~~2.2. Forbidden assignments.~~

Sometimes an activity must be prevented from occurring at certain times~~;~~ for example, swimming should not occur immediately after lunch; or practical classes should not occur in the first period of the day. Such constraints may be expressed by ~~a mapping~~:

$$\overset{\text{forbidden}}{\cancel{\text{possible}}}_0 : \text{Activity} \rightarrow \text{Period } \underline{\text{set}}$$

such that ~~possible~~ forbidden$_0$(a) gives the set of periods in which the activity a must not ~~may~~ occur in the completed timetable, i.e.

$$\forall a \qquad \text{timetable}(a) \subset \neg\overset{\text{forbidden}}{\cancel{\text{possible}}}_0(a) \qquad \cancel{(C4)}\ \cancel{B2}(C8)$$

~~These~~ The method of preassignment will often be used to ensure that the timetable has some property that cannot be expressed in terms of the other constraints described above.

2.3. Ties.

In some cases, it is desired to prevent certain related but not identical activities from occurring on the same day, for example, physics theory and physics practical, or two non-academic subjects. We may express this by a mapping:

$$\text{tie:Activity} \rightarrow \text{Activity } \underline{\text{set}}$$

which for a given activity specifies the set of <u>other</u> activities which must not occur on the same day, i.e., m

$\forall a,a',p,p'.\ a' \in tie(a)\ \&\ p \in timetable(a) \& p' \in timetable(a') \supset day\ of(p) \neq day\ of\ (p')$ ~~(C4)~~ 66

For most activities, tie(a) will be empty.

~~The conjunction of conditions (C1) to (C4) will be known as (C).~~

Unfortunately, this constraint is extremely untidy, and there is a strong temptation to ignore it, at least in the initial version of the program — a temptation to which we shall (with some misgiving) succumb.

2. ~~6. Conclusion.~~

Another useful function is:

~~tuple: l_{1} 2 .. maxlength →~~

tuple: 2 .. maxlength ~~×~~ Period → Period set

which gives for each length l and period p the ~~period set~~ period set ~~consisting of~~ consisting of a multiple period of length l starting at p.

Now we may reasonably assume that all multiperiod activities must bes spread through the week. The formal definition of the constraint on multiperiod activities ~~may now be given~~ also expresses this fact:

Let ps stand for timetable (a) ∧ periods in (d)

$\forall d.\ length(a) > 1 \supset (first(ps) \in starts\ (length\ (a))\ \&\ ps = tuple\ (length(a),\ first(ps))$ (C5) ~~(C4)~~

2.4. Preassignments.

Certain activities are specified to take place at particular times; for example, it may be necessary that one of the cooking classes takes place just before lunch, when its products are to be eaten; or perhaps a teacher is unavailable at a certain time, so that his "free" activity must be scheduled at that time. Such constraints may be expressed by a mapping:

~~preassigned~~ ~~timetable~~ timetable_0 : Activity $\rightarrow$ Period <u>set</u>

2.5 Conclusion.

The constraints given above are in practice never wholly observed by a human timetabler; and there are grounds for belief that it is often logically impossible to construct a timetable that observes all the constraints initially specified by a school. Thus in spite of the rigour with which the problem has been stated, in practice the computer cannot be expected to solve the problem as posed, but only "to do as much as it can" within the constraints, or alternatively "break as few constraints as possible". But the latter approach is not very promising, since the importance of the constraints varies from case to case, and cannot reasonably be specified in advance. "Is it more important that Mr. Jones gets his nap after lunch, or form IV should not have current affairs and swimming on the same day?" - no schoolmaster is willing to answer many hundreds of such questions in advance; but will answer a few such questions when he knows that completion of the timetable depends upon it.

3. The Timetabling Method.

The program for timetabling falls clearly into four phases:

```
timetable program:

   begin  input the data;
          ~~check consistency of data~~; carry out preassignments;
          construct the timetable;
          print the results obtained

   end
```

Of these, third phase is obviously the most difficult and should be tackled first.