15 copies Xerox please.

Timetabling for Schools.
an Exercise in Program and Data Structuring.

C. A. R. Hoare & H. C. Johnston.

1. Statement of the Problem.

<u>type</u> Item = {Jones, Smith, ..., IV, VA, VB, ..., physlab, gym, ..., projector};

lives: Item $\rightarrow$ 1 .. maxlives;

<u>type</u> Activity = {VALatin, IVphysics, ...};

<u>type</u> Period = {M1, M2, ..., F6, F7};

requirement: Activity $\rightarrow$ Item <u>set</u>; times: Activity $\rightarrow$ 1 .. size (Period);

users: Item $\rightarrow$ Activity <u>set</u>; <u>note</u> $a \in$ users$(i) \equiv i \in$ requirement(a);

timetable: Activity $\rightarrow$ Period <u>set</u>;

$\forall a.$ size (timetable(a)) = times(a) ... C1.

$\forall i, p.$ busy(i,p) = lives(i) ... C2.

<u>where</u> busy(i,p) = size $\{a \mid a \in$ users(i) & $p \in$ timetable$(a)\}$

2. Additional Constraints.

2.1. spread: Activity <u>set</u>; : Period <u>set</u>

<u>type</u> Day = {Monday, Tuesday, Wednesday, Thursday, Friday};

periodsin: Day $\rightarrow$ Period <u>set</u>;

day of: Period $\rightarrow$ Day; <u>note</u> d = dayof$(p) \equiv p \in$ periodsin(d);

$\forall a, d.$ $a \in$ spread $\supset$ size (timetable$(a) \wedge$ periodsin$(d)) \leq 1$ ... C3. $\{p \in d \mid (a,p) \in$ timetable$\}$

$\forall a.$ $a \in$ spread $\supset$ size (possdays (timetable(a))) = times(a) ... C4.

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

2.2. length: Activity $\rightarrow$ 1 .. maxlength;

starts: 2 .. maxlength $\rightarrow$ Period <u>set</u>;

tuple: 2 .. maxlength $\times$ Period $\rightarrow$ Period <u>set</u>;

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

<u>where</u> ps = timetable$(a) \wedge$ periodsin(d)

tie: Activity $\rightarrow$ Activity <u>set</u>;

$\forall a, a', p, p'.$ $a' \in$ tie(a) & $p \in$ timetable(a) & $p' \in$ timetable$(a') \supset$ dayof$(p) \neq$ dayof(p') ... C6.

timetable$_0$, forbidden: Activity $\rightarrow$ Period <u>set</u>

$\forall a.$ timetable$_0(a) \subset$ timetable$(a) \subset \neg$ forbidden(a) ... C7.

$C =_{df.}$ C1 & C2 & C3 & C4 & C5 & C6 & C7.

3. The Timetabling Method.

3.1 timetabling program:

<u>begin</u> input data; carry out preassignments; construct the timetable; print it out <u>end</u>;

progress <u>recursive</u> <u>procedure</u>:

<u>begin</u> if not consistent <u>then</u> <u>go to</u> impossible; if complete <u>then</u> <u>go to</u> printout;

<u>new</u> a: Activity, p: Period; <u>select</u> suitable (a,p);

try assignment (a,p); try cancellation (a,p);

impossible: <u>end</u>;

construct the timetable:

<u>begin</u> progress; print failure message; stop; printout: <u>end</u>;

count: <u>Integer</u> initially 0; finished: Integer <u>constant</u> size(Activity) x size (Period);

complete: count = finished

3.2. T, P: Activity → Period <u>set</u>; T := constant (empty); P := constant (full);

try assignment (a,p):

<u>begin</u> T(a) :+ p; count :+ 1; progress; count :- 1; T(a) :- p <u>end</u>;

try cancellation (a,p):

<u>begin</u> P(a) :- p; count :+ 1; progress; count :- 1; P(a) :+ p <u>end</u>;

<u>note</u> $(\text{complete} \supset T = P)$ & $\forall a \quad T(a) \subset P(a)$

3.3. N is a necessary condition for consistency means:

$\exists \text{timetable} \; (C \;\&\; \forall a\,(T(a) \subset \text{timetable}(a) \subset P(a))) \supset N$

$\forall a.$ size (T(a)) ≤ times (a) ≤ size (P(a)) ... N1.

$\forall a, i.$ busy (i,p) ≤ lines (i) ≤ possbusy (i,p) ... N2.

<u>where</u> busy (i,p) = size ({a | a ∈ users (i) & p ∈ T(a)})

possbusy (i,p) = size ({a | a ∈ users (i) & p ∈ P(a)})

$\forall a, d.$ a ∈ spread ⊃ size (T(a) ∧ periods in (d)) ≤ 1 ... N3.

$\forall a, d.$ a ∈ spread ∨ length(a) > 1 ⊃

possdays(T(a)) x length (a) ≤ times (a) ≤ possdays (P(a)) x length (a) ... N4.

$\forall a, d.$ length (a) > 1 ⊃ times (a) ≤ size ({d | posstuples (a,d) ≠ empty}) x length (a) ... N5.

3.4 FA (a,p) is sufficient for forcing assignment means: $FA(a,p) \supset \neg N^{P}_{(P \frown a: P(a)-p)}$

FC (a,p) is sufficient for forcing cancellation means: $FC(a,p) \supset \neg N^{T}_{(T \frown a: T(a)+p)}$

forced assign: Activity → Period <u>set</u> <u>initially</u> timetable_0;

forced cancel: Activity → Period <u>set</u> <u>initially</u> forbidden;

select suitable (a,p): {select forced (a,p); select unforced (a,p)}

select forced (a,p):

<u>begin</u> if forced assign ≠ empty <u>then</u> {(a,p) <u>from</u> forced assign; try assignment (a,p); <u>go to</u> impossible}

<u>else</u> if forced cancel ≠ empty <u>then</u> {(a,p) <u>from</u> forced cancel; try cancellation (a,p); <u>go to</u> impossible}

<u>end</u>;

Assuming $p \in P(a) - T(a)$:

size $(P(a))$ = times (a) ... FA1.

$\exists i.$ $i \in$ requirement(a) & possbusy(i,p) = lives(i) ... FA2.

(no corresponding condition) ... FA3.

daydetermined(a,P) & size $(P(a) \wedge$ periodsin (dayof(p))) = length(a) ... FA4.

where daydetermined(a,X) = $(a \in$ spread $\vee$ length$(a) > 1)$ & times(a) = possdays $(X(a)) \times$ length(a).

length$(a) > 1$ & $p \in \bigcap$ posstuples $(a$, dayof$(p))$ & daydet$(a$, dayof$(p))$ ... FA5.

where daydet(a,d) = (daydetermined(a,P) $\vee$ $T(a) \wedge$ periods in $(d) \neq$ empty)

times(a) = size $(T(a))$ ... FC1.

$\exists i.$ $i \in$ requirement(a) & lives(i) = busy(i,p) ... FC2.

$\exists p'.$ $a \in$ spread & $p' \in T(a)$ & $p \in$ periodsin (dayof(p')) ... FC3.

daydetermined(a,T) & dayof(p) $\tilde{\in}$ possdays $(T(a))$ ... FC4.

length$(a) > 1$ & $p \tilde{\in} \bigcup$ posstuples $(a$, dayof$(p))$ ... FC5.

3.5. stiff: Activity $\rightarrow$ Period set   Decision set

select unforced(a,p):

if stiff $\neq$ empty then{(a,p) from stiff;   find $(a,p) \in$ stiff s.t. $(a,p) \tilde{\in}$ ass $\vee$ conc

while stiff $\neq$ empty & $p \tilde{\in} P(a)-T(a)$ do (a,p) from stiff;   else select nonstiff(a,p)

if $p \tilde{\in} P(a)-T(a)$ then select nonstiff(a,p)}

else select nonstiff(a,p)

sorted: Activity sequence

select nonstiff(a,p):   select nonstiff(a,p); find $a \in$ sorted

repeat {a := head (sorted); if $P(a)-T(a)$ = empty then sorted := tail(sorted) else let $p \in P(a)-T(a)$}

until $p \in P(a)-T(a)$

Assuming $p \in P(a)-T(a)$

size $(P(a))$ − times(a) = 1 ...ST1

$\exists i.$ $i \in$ requirement (a) & possbusy (i,p) − lives(i) = 1 ...ST2

$\exists d.$ $a \in$ spread & size $(P(a) \wedge$ periodsin (dayof(p))) = if daydetermined(a,P) then 2 else 1 ...ST3

$\exists d.$ length$(a) > 1$ & ($T(a) \wedge$ periods in (dayof (p)) $\neq$ empty

$\vee$ daydetermined (a,P) $\wedge$ size$(P(a) \wedge$ periodsin (dayof(p))) − length(a) = 1) ... ST4

3.6 Premature termination.

countmax: Integer initially 0; limit: Integer;

after count:+1 do if count > countmax then countmax := count;

after count:−1 do if count − limit then {print out T; stop}

Exercise: correct the errors in the formulation given above.

4. The Program.

```
check consistency:
begin if p ∈ T(a) then note the most recent decision was T(a):+p;
    begin if p ∉ P(a) then go to impossible;
        case times(a) - size(T(a)) of {<0: go to impossible;                                        ... N1
                                       =0: for p' ∈ P(a)-T(a) do forcedcancel(a):+p'};               ... FC1

        for i ∈ requirement(a) do case lives(i) - busy(i,p) of
                                  {<0: go to impossible;                                            ... N2
                                   =0: for a' ∈ users(i) do if p ∈ P(a)-T(a) then forcedcancel(a'):+p}; FC2

        if a ∈ spread then {if size(T(a) ∧ periodsin(dayof(p))) > 1 then go to impossible           ... N3
                            else for p' ∈ periodsin(dayof(p)) do if p' ≠ p then forcedcancel(a):+p}; ... FC3
        if a ∈ spread ∨ length(a) > 1 then case times(a) ÷ length(a) - size(possdays(T(a))) of
                                  {<0: go to impossible;                                            ... N4
                                   =0: for d ∈ ¬ possdays(T(a)) do
                                          for p' ∈ P(a) ∧ periodsin(d) do
                                              forcedcancel(a):+p'};                                 ... FC4
        if length(a) > 1 & T(a) ∧ periodsin(dayof(p)) ≠ empty then
                     for p' ∈ P(a) ∧ periodsin(dayof(p)) - T(a) do stiff(a):+p'                      ... ST4
    end
else begin note the most recent decision was P(a):-p;
        case size(P(a)) - times(a) of {<0: go to impossible;                                        ... N1
                                       =0: for p' ∈ P(a)-T(a) do forcedassign(a):+p';                ... FA1
                                       =1: for p' ∈ P(a)-T(a) do stiff(a):+p'};                      ST1
        for i ∈ requirement(a) do case possbusy(i,p) - lives(i) of
                                  {<0: go to impossible;                                            ... N2
                                   =0: for a' ∈ users(i) do if p ∈ P(a)-T(a) then forcedassign(a'):+p; ... FA2
                                   =1  for a' ∈ users(i) do if p ∈ P(a)-T(a) then stiff(a):+p};       ... ST2
        if a ∈ spread & size(P(a) ∧ periodsin(dayof(p))) = 1 then stiff(a):+first(P(a) ∧ periodsin(dayof(p))); ... ST3
        if P(a) ∧ periodsin(dayof(p)) = empty & (a ∈ spread ∨ length(a) > 1) then
          case size(possdays(P(a))) × length(a) - times(a) of
            begin <0: go to impossible;                                                             ... N4
                  =0: for d ∈ possdays(P(a)) do note daydetermined(a,P);
                          case size(P(a) ∧ periodsin(d)) - length(a) of
                            {<0: go to impossible;
                             =0: for p' ∈ P(a) ∧ periodsin(d) - T(a) do forcedassign(a):+p';         ... FA4
                             =1: for p' ∈ P(a) ∧ periodsin(d) - T(a) do stiff:+p'}.                  ... ST4
            end.
    end;
```

(continued on next page..)

```
progress (a: Activity; p: Period) recursive procedure
  begin if count > 0 then check consistency;
        if count = finished then go to printout;
        new a: Activity; p: Period;
        cont' procedure
           begin count:+1; if count > countmax then countmax := count;
                 progress (a,p);
                 count:-1; if countmax - count > limit then {print T; stop}
           end cont;
        try assignment procedure {T(a):+p; for i ∈ requirement(a) do busy(i,p):+1; cont;
                                  for i ∈ requirement(a) do busy(i,p):-1; T(a):-p; stiff(a):+p};

        if forcedassign ≠ empty then {(a,p) from forcedassign; try assignment; go to impossible};
        if forced cancel ≠ empty then {(a,p) from forced cancel; try cancellation; go to impossible};
        if stiff ≠ empty then {(a,p) from stiff;
        while stiff ≠ empty & p ε̃ P(a) - T(a) do (a,p) from stiff;
        if p ε̃ P(a) - T(a) then select nonstiff}
                          else select nonstiff
         where select nonstiff procedure
              repeat {a := head (sorted); if P(a) - T(a) = empty then sorted := tail(sorted) else p := first(P(a) - T(a))}
              until p ∈ P(a) - T(a);
        try assignment; forcedassign := forced cancel := empty; try cancellation;
impossible:
 end progress;

  try cancellation procedure {P(a):-p; for i ∈ requirement(a) do possbusy(i,p):-1; cont;
                              for i ∈ requirement(a) do possbusy(i,p):+1; P(a):+p; stiff(a):+p};
```

Exercise: adapt this program to take into account the effect of ties.

```
if length(a) > 1 then
   begin new intersection := periods in (day of (p)); new union := empty;
      for p' ∈ starts (length(a)) ∧ periods in (day of (p)) do
         { new ps := tuple (p', length(a));
              if T(a) ⊂ ps ⊂ P(a) then {intersection :∧ ps; union :∨ ps}
         };
      if daydet (a, dayof(p)) then for p' ∈ intersection − T(a) do  forcedassign(a) :+ p';      FA5.
      for p' ∈ P(a) − union do  forced cancel(a) :+ p'                                           FC5.
```

5. Data Representation.

Available: 12K 24-bit words for data, leaving 4K for program.

type Poolpointer = 1..2047; Pool: Poolpointer → Word;

A: Activity → (times: 0..31; length: 0..3; spread: {yes, no};
requirement: (first: 0..255; if first = 0 then long: Poolpointer
else rest: 1..4 → 0..255));

I: Item → (lives: 1..63; users: Poolpointer if lives > 1 then (pbptr: Poolpointer;
busy: 1.. size(Period) → 0..63)
else 1.. size (Period) → (busy: Boolean; possbusy: 1..31))

T, P: 1.. size(Activity) → Period set;

F: 1..300 → (a: Activity; p: Period; d: {assign, cancel});
forcetop initially 0; stifftop initially 301;

Sorted: Activity → [1..2048];

periods in: Day → Periodset; dayof: Period → Day; periods in day of: Period → Period set; tuple

Stack: 1..100 → (a: Activity; p: Period; f: {forced, unforced})

Total			
Pool	2048		2000
A	750×2		1500
I	$250 \times (3 + \frac{48}{4})$		3750
T, P	2 × 750 × 2		3000
F	300		300
Sorted	$\frac{750}{2}$		375
periods in, etc.	12 + 48 + 96	say	200
Stack	300		300
			11 25

In the pool long: 40 × 10 = 400
users: $250 \times \frac{10}{2}$ = 1250
pbtr:

Exercise: code this program in the language of your choice.