Appendix B

Extracts of definitive scripts for timetabling model

by EM group members

/*timetable.e*/

```
include("util.e");

/* ============
   Input data
   ============ */

class = ['x','y','z'];
staff = ['X','Y'];
slots = [1,2,3,4,5,7,8,10];

AVSTAFF=[['X',[2,3,5]],['Y',[3,5,6]]];
SPSTAFF=[['X',['x','y']],['Y',['z','x']]];
avstud=[['x',[1,2,3]],['y',[2,3,4]],['z',[3,4,5]]];

TT=[['x',2],['y',3],['x',4]];
ASS=[['x','X'],['y','Y']];

/* ======================================
   functions that contribute to messages
   ====================================== */

func endmsg /* $1=student */
{
   return ".^. for student " // $1 // "\n";
}

func endMSG /* $1=staff */
{
   return ".^. for staff " // $1 // "\n";
}

func single_slot /* $1 = slx, $2 = student */
{
return ($1# != 1) ? $2 // " has " // tostr($1#) // " slots\n" : "";
}

func avail_x /* $1 = student, $2 = slx, $3 = avx */
{
return (meet($2,$3)!=$2) ?
  $1 // " unavailable"
// " slx is " // tostr($2)
// " avx is " // tostr($3) // "\n"
:
"";
}
```

```
func has_assessor /* $1 = student, $2 = asx */
{
auto n;
n = $2#;
return ((n>2)||(n==0)) ?
$1 // " has " // tostr(n) // " assessors\n"
:
"";
}


func avail_X /* $1 = staffmem, $2 = SLX, $3 = AVX */
{
return (meet($2, $3) != $2) ?
  $1 // " unavailable"
// " SLX is " // tostr($2)
// " AVX is " // tostr($1) // "\n"
:
"";
}

func suit_X /* $1 = staffmem, $2 = ASX, $3 = SPX */
{
return (meet($2, $3) != $2) ?
  $1 // " unsuitable"
// " ASX = " // tostr($2)
// " SPX = " // tostr($3) // "\n"
:
"";
}

func count { return $1#; }

/* need to do something about it */

header =
"\n*******\n" //
"TT is " // tostr(TT) // "\n" //
"ASS is " // tostr(ASS) // "\n" //
"AVSTAFF is " // tostr(AVSTAFF) // "\n" //
"avstud is " // tostr(avstud) // "\n" //
"SPSTAFF is " // tostr(SPSTAFF) // "\n" //
"*****\n";

func not_dblebk /* $1 = TT */
{
   auto i,j,l,overbook;
   overbook = "";
   for (i=1; i<=20; i++) {
l = [];
for (j=1; j<=$1#; j++) {
   if ($1[j][2]==i) {
append l, $1[j][1];
   }
}
if (l#>1) overbook = overbook // "Slot " // tostr(i) // " overbooked\n";
   }
   return overbook;
}
```

```
/* ======================
   elementary functions
  ====================== */

func meet
{
auto i,j,st;
st=[];
for (i=1; i<=($1)#; i++) {
for (j=1; j<=($2)#; j++) {
if ($1[i]==$2[j]) {
append st, $1[i];
}
}
}
return(st);
}

func proj2
{
auto i,l;
l = [];
for (i=1; i<=($1)#; i++) {
if (($1)[i][1]==$2) {append l, ($1)[i][2];}
}
return(l);
}

func proj2_1 { return proj2($1, $2)[1]; }

func proj1
{
auto i,l;
l = [];
for (i=1; i<=($1)#; i++) {
if (($1)[i][2]==$2) {append l, ($1)[i][1];}
}
return(l);
}

func join
{
auto i,j;
l=[];
for (i=1; i<=($1)#; i++) {
for (j=1; j<=($2)#; j++) {
if (($1)[i][1]==($2)[j][1]) {
append l, [($1)[i][2],($2)[j][2]];
}
}
}
return (l);
}
```

```
/* ==============
   definitions
  ============== */

NPX is map(count, ASX);
asx is map(proj2, [ASS], class);
slx is map(proj2, [TT], class);
SLX is map(proj2, [join(ASS,TT)], staff);
ASX is map(proj1, [ASS], staff);
AVX is map(proj2_1, [AVSTAFF], staff);
SPX is map(proj2_1, [SPSTAFF], staff);
avx is map(proj2_1, [avstud], class);

e_student is map(endmsg, class);
e_staff is map(endMSG, staff);
s_s is combine(single_slot, [slx, class]);
a_x is combine(avail_x, [class, slx, avx]);
h_a is combine(has_assessor, [class, asx]);
a_X is combine(suit_X, [staff, SLX, AVX]);
s_X is combine(suit_X, [staff, ASX, SPX]);
msg is combine(strcat, [s_s, a_x, h_a, e_student]);
MSG is combine(strcat, [a_X, s_X, e_staff]);

/* ==============
   show result
  ============== */

proc showheader : TT, ASS, AVSTAFF, avstud, SPSTAFF
{
writeln(header);
writeln(not_dblebk(TT));
}

proc showresult : class, staff
{
map(writeln, msg);
map(writeln, MSG);
}
```

/*timetable.s*/

```
%scout
window dataW;
string header;
dataW = {
frame:([{1.c, 1.r}, 8, 50]),
string:header,
border:1
};

window dbW;
string ndb;
dbW = {
frame:([{1.c, 10.r}, 5, 30]),
string:ndb,
border:1
};

window studentW;
string studmsg;
studentW = {
frame:([{1.c, 15.r}, 5, 30]),
string:studmsg,
border:1
};

window staffW;
string staffmsg;
staffW = {
frame:([{1.c, 20.r}, 5, 30]),
string:staffmsg,
border:1
};

screen = < dataW / dbW / studentW / staffW >;

integer student, staffmem;

%eden
include("timetable.e");

ndb is not_dblebk(TT);
studmsg is msg[locate(class, student)];
staffmsg is MSG[locate(staff, staffmem)];
func locate /* list, element */
{
auto i;
for (i = 1; $1[i] != $2; i++);
return i;
}

student = 'x';
staffmem = 'X';
%end
```