Under consideration for publication in Formal Aspects of Computing

# Model Checking Learning Agent Systems using Promela with Embedded C code and Abstraction

Ryan Kirwan[1], Alice Miller[1] and Bernd Porr[2]
[1]School of Computing Science, University of Glasgow, Glasgow, UK
[2]School of Electronic Engineering, University of Glasgow, Glasgow, UK

This file contains Promela models for the paper submitted to Journal of Systems and Software. All files can be downloaded from:

`http://www.dcs.gla.ac.uk/people/personal/alice/appendices/publications/FAC2016_code/`

## 1. Explicit model

To verify the Explicit model you need the main Promela file, `ExplWithCcodeM6.pml`, and files `newExMo.h` and `newExMoInLines.txt` containing C functions and inline functions respectively. To run verifications, use the shell script `verifyExplModel6.sh`. The file `explObGenV211.c` is used to generate random environments.

### 1.1. Main Promela file

This is the main Promela file, namely `ExplWithCcodeM6.pml`. To run properties P1: and P2: from the paper, select the appropriate ltl statement at the end of the main Promela file. Adjust the number of obstacles by amending the OBMAX definition line (line 4). The environment should be selected by amending the `SETENV` call in the `init` statement.

```
/*Explicit Model: Using Functions (Macros)*/

c_decl { #include <math.h> }
#include "newExMo.h"
#include "newExMoInLines.txt"
#define OBMAX 3
#define MAX_OMEGAD 1

/*Define a polar coordinate to be a distance and angle from origin (pole)*/
/*(Pole: centre of the environment. Polar axis: vertical line directed north.)*/
typedef polarCoord {int d; int a};

/*Setting the C_Track variables*/
c_track"&x" "sizeof(double)"
c_track "&prevX" "sizeof(double)"
```

*Correspondence and offprint requests to*: Alice Miller, School of Computing Science, University of Glasgow, Glasgow, UK. e-mail: alice.miller@glasgow.ac.uk

```
c_track "&roboA" "sizeof(double)"
c_track "&enviD" "sizeof(long double)"
c_track "&enviA" "sizeof(long double)"

c_track "&obD" "sizeof(double)"
c_track "&obA" "sizeof(double)"
c_track "&relD" "sizeof(double)"
c_track "&relA" "sizeof(double)"

c_track "&obDist" "sizeof(int)"
c_track "&obAng" "sizeof(int)"
c_track "&moveDist" "sizeof(int)"
c_track "&inCone" "sizeof(int)"

/*Array of obstacles in the fixed environment*/
polarCoord arrObs[OBMAX];

/*Robot is initiated in the centre of the environment*/
int roboAng;
int enviDist;
int enviAng ;
int omegaD;
int sig;
int prevSig;
byte doWrap;
byte headOn;
int relDist;

proctype robot() {
do
:: (doWrap==0) ->d_step{SCAN_APPROACHING_OBS();
RESPOND();
MOVE_ROBOT();
HEAD_ON();
};
:: (doWrap==1) ->d_step{ WRAP();
};
od;
};


init {
d_step{
/* Set up the polar coordinates of the obstacles - fixed for model */



          SETENV1(); /* amend per environment */
          roboAng = 0;
  enviDist = 0;
  enviAng = 0;
  omegaD = 0;
  doWrap = 0;
  sig = 0;
  prevSig = 0;
  headOn = 0;
  /*Relative distance should be set to outwith the "cone
of influence" initially to stop property violations.*/
  relDist = 200;

  c_code{
roboA = 0;
enviA = 0;
enviD = 0;
};
};
atomic{ run robot();};
};


#define p ((sig<=-6)||(sig>=6))
#define d ((prevSig!=0) && (prevSig>-6) && (prevSig<6) )
#define a (omegaD <= MAX_OMEGAD)


/* select property */
/*ltl p1 { <>[](p->!d)}*/
ltl p2 { [] a }
```

## 1.2. C function file

This is file `newExMo.h`. It contains C code defining constants and functions, as desribed in the paper. The version below has all debugging code removed, see the actual code for this.

```
c_code {
#define PI 3.141592653
#define DEG 0.01745329
#define ROBORAD 20
#define OBRAD 10
#define AAL 11
#define AAR 69
#define ADIST 8
#define SENLEN 80
#define LAMBDA 1
#define OMEGAP 15
#define ROBOMOVE 1
#define ENVIRAD 200
#define ENVIDIAM 400
#define ANTLEN 60

#define CRASHANGFROMCENTRE 9.825648155

        //GLOBALS//


double x = 0;
double prevX = 0;

double roboA = 0;
long double enviA = 0;
long double enviD = 0;

double obD = 0;
double obA = 0;
double relD = 0;
double relA = 0;

int moveDist = 1;
int obDist = 0;
int obAng = 0;
int inCone = 0;


int testFlag = 0;


// fprintf(fp, "FINAL: curAng: %d. curDist: %d \n", obAng, obDist);

// fclose(fp);

///////////
```

```
};


#define MOVE_FORWARD() { \
/*Declare Locals*/ \
if (now.relDist > 30) { \
long double oZ = 0; \
long double nZ = 0; \
long double fZ = 0; \
long double lOrg = 0; \
long double hOrg = 0; \
long double lNew = 0; \
long double hNew = 0; \
long double lFin = 0; \
long double hFin = 0; \
int oFR = 0; \
int oFU = 0; \
int nFR = 0; \
int nFU = 0; \
\


oZ = fmodl(enviA, (long double)90); \
if ((enviA == 90) || (enviA ==270)) {lOrg = enviD; hOrg = 0; } \
else if ((enviA == 0) || (enviA == 180)) {lOrg = 0; hOrg = enviD; } \
else { \
if ((enviA <=90) || ((enviA >=180)&&(enviA<=270))) { \
lOrg = (sin(oZ*DEG))*enviD; \
hOrg = (cos(oZ*DEG))*enviD; \
} else { \
hOrg = (sin(oZ*DEG))*enviD; \
lOrg = (cos(oZ*DEG))*enviD; \
} \
} \
\
nZ = fmod(roboA, 90.00);  \
if ((roboA == 90) || (roboA ==270)) {lNew = moveDist; hNew = 0; } \
else if ((roboA == 0) || (roboA == 180)) {lNew = 0; hNew = moveDist; } \
else { \
if ((roboA<90) || ((roboA>180)&&(roboA<270))) { \
lNew = (sin(nZ*DEG))*moveDist; \
hNew = (cos(nZ*DEG))*moveDist; \
} else { \
hNew = (sin(nZ*DEG))*moveDist; \
lNew = (cos(nZ*DEG))*moveDist; \
} \
} \

if ((enviA<180)&&(roboA>180) || (enviA>180)&&(roboA<180)) { lFin = fabs(lOrg - lNew);} \
  else {  lFin = lOrg + lNew;} \
if ( (((enviA<90)||(enviA>270))&&((roboA>90)&&(roboA<270))) ||
              (((enviA>90)&&(enviA<270))&&((roboA<90)||(roboA>270))) ) {  \
hFin = fabs(hOrg - hNew); \
} else { hFin = hOrg + hNew; } \
if ((hFin!=0)&&(lFin!=0)) { fZ = (atan(hFin/lFin)*(180/PI)); } \
else { fZ = 0;} \
enviD = sqrt((lFin*lFin)+(hFin*hFin)); \
/*Finding the quadrant of the original position of the robot.*/ \

if ((enviA >=0) && (enviA <90))  { oFR = 1; oFU = 1;} \
else if ((enviA >= 90) && (enviA <180))  { oFR = 1; oFU = 0;} \
else if ((enviA >= 180) && (enviA <270)) { oFR = 0; oFU = 0;} \
else if ((enviA >= 270) && (enviA <360)) { oFR = 0; oFU = 1;} \
else {if (debug ==1){ Printf("FECKTAL ERROR with enviA = %Lf. \n", enviA);} oFR = 0; oFU = 0;} \
nFR = oFR; \
nFU = oFU; \
/*Finding the quadrant of the original position of the robot.*/ \
if ((oFR==1) && (oFU==1)) {  \
if ((roboA>=180) && (roboA<360) && (lNew > lOrg)) { nFR = 0;} \
if ((roboA>=90) && (roboA<270) && (hNew > hOrg))  { nFU = 0;} \
} \
else if ((oFR==1) && (oFU==0)) {  \
if ((roboA>=180) && (roboA<360) && (lNew > lOrg)) { nFR = 0;} \
if ( (((roboA>=270)&&(roboA<360))||((roboA>=0)&&(roboA<90))) && (hNew > hOrg)) { nFU = 1;} \
} \
else if ((oFR==0) && (oFU==0)) {  \
if ((roboA>=0) && (roboA<180) && (lNew > lOrg))   { nFR = 1;} \
if ( (((roboA>=270)&&(roboA<360))||((roboA>=0)&&(roboA<90))) && (hNew > hOrg)) { nFU = 1;} \
} \
else if ((oFR==0) && (oFU==1)) {  \
if ((roboA>=0) && (roboA<180) && (lNew > lOrg))   { nFR = 1;} \
if ((roboA>=90) && (roboA<270) && (hNew > hOrg))  { nFU = 0;} \
} \


/*Many catches for when the movement is along the axis lines/opposing
them,
         or when both are the same.*/ \
if (roboA==enviA) { enviA = roboA;} \
else if ((roboA==180)&&(enviA==0)&&(hNew>hOrg)) {enviA = 180;} \
else if ((roboA==180)&&(enviA==0)&&(hNew < hOrg)) {enviA = 0;} \
else if ((roboA==180)&&(enviA==0)&&(hNew == hOrg)) {enviA = 0;} \
else if ((roboA==0)&&(enviA==180)&&(hNew>hOrg)) {enviA = 0;} \
else if ((roboA==0)&&(enviA==180)&&(hNew < hOrg)) {enviA = 180;} \
else if ((roboA==0)&&(enviA==180)&&(hNew == hOrg)) {enviA = 0;} \
else if ((roboA==90)&&(enviA==270)&&(lNew>lOrg)) {enviA = 90;} \
else if ((roboA==90)&&(enviA==270)&&(lNew < lOrg)) {enviA = 270;} \
else if ((roboA==90)&&(enviA==270)&&(lNew == lOrg)) {enviA = 0;} \
else if ((roboA==270)&&(enviA==90)&&(lNew>lOrg)) {enviA = 270;} \
else if ((roboA==270)&&(enviA==90)&&(lNew < lOrg)) {enviA = 90;} \
else if ((roboA==270)&&(enviA==90)&&(lNew == lOrg)) {enviA = 0;} \
else if ((nFR==1)&&(nFU==1)) { enviA = 90 - fZ;} \
else if ((nFR==1)&&(nFU==0)) { enviA = 90 + fZ;} \
else if ((nFR==0)&&(nFU==0)) { enviA = 270 - fZ;} \
else if ((nFR==0)&&(nFU==1)) { enviA = 270 + fZ;} \
if (enviA>=360) {now.enviAng = 0;} \
else {now.enviAng = ((int)(2*enviA)) - ((int)enviA);} \
enviD = ((int)(2*enviD)) - ((int)enviD); \

/*Don't want to wrap from a wrap.*/ \
if ((enviD>=200) && (now.doWrap==0)) { enviD = 200; now.doWrap=1;} \
now.enviDist = (int)enviD; \

} \
};


#define MOVE_ROBOT() c_code { \

\
/*Setting the roboA and the moveDist before MOVE_FORWARD() is called*/\
roboA = (double) now.roboAng; \
moveDist = ROBOMOVE; \
MOVE_FORWARD(); \
};


#define GET_OB_REL_TO_ROBOT() {\
/*Declare Locals*/ \
long double oZ = 0; \
long double nZ = 0; \
long double fZ = 0; \
long double lOrg = 0; \
long double hOrg = 0; \
long double lNew = 0; \
long double hNew = 0; \
long double lFin = 0; \
long double hFin = 0; \
int furtherRight = 0; \
int furtherUp = 0; \
double roboED = 0; \
double roboEA = 0; \
double roboFarAntiClock = 0; \
double relAN = 0; \
\
roboA = (double) now.roboAng; \
roboED = (double)enviD; /*(double) now.enviDist;*/ \
roboEA = (double)enviA; /*(double) now.enviAng;*/ \
obD = (double)obDist; \
obA = (double)obAng; \
oZ = ((int)roboEA)%90; \
if ((roboEA == 90) || (roboEA ==270)) {lOrg = roboED; hOrg = 0; } \
else if ((roboEA == 0) || (roboEA == 180)) {lOrg = 0; hOrg = roboED; } \
else { \
if ((roboEA <=90) || ((roboEA >=180)&&(roboEA<=270))) { \
lOrg = (sin(oZ*DEG))*roboED; \
hOrg = (cos(oZ*DEG))*roboED; \
} else { \
hOrg = (sin(oZ*DEG))*roboED; \
lOrg = (cos(oZ*DEG))*roboED; \
} \
} \
nZ = ((int)obA)%90; \
if ((obA == 90) || (obA ==270)) {lNew = obD; hNew = 0; } \
else if ((obA == 0) || (obA == 180)) {lNew = 0; hNew = obD; } \
else { \
if ((obA<90) || ((obA>180)&&(obA<270))) { \
lNew = (sin(nZ*DEG))*obD; \
hNew = (cos(nZ*DEG))*obD; \
} else { \
hNew = (sin(nZ*DEG))*obD; \
lNew = (cos(nZ*DEG))*obD; \
} \
} \
if ((roboEA<180)&&(obA>180) || (roboEA>180)&&(obA<180)) { lFin = lOrg + lNew;} \
  else {  lFin = fabs(lOrg - lNew);} \
if ( (((roboEA<90)||(roboEA>270))&&((obA>90)&&(obA<270)))
                || (((roboEA>90)&&(roboEA<270))&&((obA<90)||(obA>270))) ) { \
hFin = hOrg + hNew; \
} else { hFin = fabs(hOrg - hNew);} \
fZ = (atan(hFin/lFin)*(180/PI)); \
if ((roboEA<=180)&&(obA<=180)) { if (lOrg > lNew) {
    furtherRight = 1;}
               else { furtherRight = 0;}} \
else if ((roboEA>=180)&&(obA>=180)) { if (lOrg > lNew) {
    furtherRight = 0;}
             else { furtherRight = 1;}} \
else if ((roboEA>180)&&(obA<180)) { furtherRight = 0;} \
else if ((roboEA<180)&&(obA>180)) { furtherRight = 1;} \
else {furtherRight = 0; furtherUp = 0;} \
if (((roboEA<=90)||(roboEA>=270))&&((obA<=90)||(obA>=270))){
  if(hOrg > hNew) { furtherUp=1;}
            else { furtherUp=0;}} \
else if
(((roboEA>=90)&&(roboEA<=270))&&((obA>=90)&&(obA<=270))) {
  if(hOrg>hNew) { furtherUp=0;}
               else { furtherUp=1;}} \
else if (((roboEA<=90)||(roboEA>=270)) && ((obA>=90)&&(obA<=270))) { furtherUp = 1;} \
else if (((roboEA>=90)&&(roboEA<=270)) && ((obA<=90)||(obA>=270))) { furtherUp = 0;} \
if ((furtherRight==1) && (furtherUp==1)) { relAN = 270 - fZ;} \
else if ((furtherRight==1) && (furtherUp==0)) { relAN = 270 + fZ;} \
else if ((furtherRight==0) && (furtherUp==1)) { relAN = 90 + fZ;} \
else if ((furtherRight==0) && (furtherUp==0)) { relAN = 90 - fZ;} \
roboFarAntiClock = ((((int)roboA) - 40)+360)%360; \
relAN = ((int)(2*relAN)) - ((int)relAN); \
relA = (((int)(relAN - roboFarAntiClock))+360)%360; \
relD = sqrt((hFin*hFin) + (lFin*lFin)); \
if ((relA <= 80)&&(relD <= 90)) { inCone = 1; } \
else { inCone = 0;} \
now.relDist = ((int)(2*relD)) - ((int)relD); \
/*now.relAng = (int)relA;*/ \
};


#define RESPOND() c_code { \
prevX = x; \
now.prevSig = now.sig; \
if (inCone==1) { \
RESPOND_TO_OB_BY_TURNING(); \
now.sig = (int)(fabs(x)); \
} else { \
          /*If not inCone then we need to reset the signals*/ \
x = 0; \
now.sig = 0; \
} \
Printf("SEND IN THE SIGNAL: now.sig %d, now.prevSig %d. \n", now.sig, now.prevSig); \
}; \


#define RESPOND_TO_OB_BY_TURNING() { \
/*Declare Locals*/ \
double theta = 0; \
double opp = 0; \
double adj = 0; \
double sAdj = 0; \
double sHyp = 0; \
int i = 0; \
signed int flag = 0; \
double contactPoint = 0; \
double oAMAX = 0; \
double lowR = 0; \
double highR = 0; \
double dOmegaD = 0; \
\
roboA = (double)now.roboAng; \
dOmegaD = (double)now.omegaD; \
oAMAX = (atan((OBRAD/relD)))*(180/PI); \
lowR = (relA - oAMAX); \
if (lowR < 0) { lowR = 0;} \
highR = (relA + oAMAX); \
if (highR > 80) { highR = 80;} \
if ( (AAL >= lowR) & (AAL <= highR) ) { theta = fabs(AAL - relA); flag = 1;} \
else if ( (AAR >= lowR) & (AAR <= highR) ) { theta = fabs(AAR - relA); flag = -1;} \
else {flag = 0;} \
opp = relD*(sin(theta*(PI/180)));\
if ((flag != 0 ) && (opp <= OBRAD)) { \
adj = sqrt((relD*relD) - (opp*opp)); \
if (adj <= 80) { \
for (i=0; i<=10; i++) { \
sAdj = OBRAD - i; \
sHyp = sqrt((sAdj*sAdj) + (opp*opp));  \
if ((sHyp != 0) & (sHyp <= OBRAD)) {  \
contactPoint = adj - sAdj; \
x = (flag)*(ADIST - ( ( contactPoint-(((int)contactPoint)%10) )/10 )); \
x = ((int)(2*x)) - ((int)x); \
break; \
} \
} \
} \
else { x = 0;} \
} \
else { x = 0;}  \
if ((x == 0) && (now.relDist <= 30)) { /*This is a crash*/ CRASH();} \
else if (fabs(x) > 5) { roboA = (((int)(roboA + (x*OMEGAP)))+360)%360; LEARN_NOW();} \
else { roboA = (((int)(roboA + (x*dOmegaD)))+360)%360;} \
if (roboA>=360) { roboA=0;} \
now.roboAng = (int)roboA; \
};


#define LEARN_NOW() { \
if ( (prevX>=(-5)) && (prevX<=5) && (prevX!=0) ) { \
now.omegaD = now.omegaD + LAMBDA;\
} \
};


#define CRASH() { \
/*Declare Locals*/ \
double oZ = 0; \
double oOpp = 0; \
double oAdj = 0; \
double oHyp = 0; \
double nZ = 0; \
double nOpp = 0; \
double nAdj = 0; \
\
/*If we crash with no proximal impact --We are inCone*/ \
if ((relA==40) && (relD==30)) { \
now.headOn = 1; /*Promela macro? Fixed Angle is 9.825648155*/ \
} else if ((relA>=30)&&(relA<40)) { \
oZ = relA - 11; \
oHyp = 30; \
oOpp = sin(oZ*DEG)*oHyp; \
oAdj = sqrt((oHyp*oHyp)-(oOpp*oOpp)); \
nOpp = oOpp - OBRAD; \
nAdj = oAdj; \
nZ = (atan(nOpp/nAdj))*(180/PI); \
x = 6; \
roboA = ((int)(roboA + (x*OMEGAP) + nZ))%360; \
} else if ((relA<=50)&&(relA>40)) { \
oZ = (relA - 40) - 11; \
oHyp = 30; \
oOpp = sin(oZ*DEG)*oHyp; \
oAdj = sqrt((oHyp*oHyp)-(oOpp*oOpp)); \
nOpp = oOpp - OBRAD; \
nAdj = oAdj; \
nZ = (atan(nOpp/nAdj))*(180/PI); \
x = (-6); \
roboA = (((int)(roboA + (x*OMEGAP) - nZ))+360)%360; \
} \

now.roboAng = ((int)(2*roboA)) - ((int)roboA); \
Printf("CRASH():nZ %f, roboA %f, now.roboAng %d. \n", nZ, roboA, now.roboAng); \
};


#define WRAP() c_code { \
/*Declare Locals*/ \
int l = 0; \
long double orgEnviA = 0; \
long double orgEnviD = 0; \
int oppAng = 0; \
\
roboA = (double)now.roboAng; /*We don't want to change roboAng.*/ \
moveDist = ENVIDIAM; \
oppAng = ((int)(roboA+180))%360; \
roboA = oppAng; /*for MF function.*/ \
orgEnviA = enviA; \
orgEnviD = enviD; \
\

MOVE_FORWARD(); \
for (l=399; l--; l>=0) { \
\
if (enviD > 200) { \
/*Reset the envi Values.*/ \
enviA = orgEnviA; \
enviD = orgEnviD; \
moveDist = 1; \
MOVE_FORWARD(); \
} else {break;} \
} \
roboA = (double)now.roboAng; /*Reset the roboAng.*/ \
now.doWrap = 0; \
};


#define TRIVIAL_WRAP() c_code { \
/*Printf("DOES THIS WORK!? \n");*/ \
now.enviAng = (now.enviAng + 180)%360; \
now.enviDist = 200; \
enviA = ((int)(2*enviA)) - ((int)enviA); \
enviA = ((int)enviA + 180)%360; \
enviD = 200; \

now.doWrap = 0; \
};

#define SCAN_APPROACHING_OBS() c_code { \
/*Declare Locals*/ \
int j = 0; \
\
inCone = 0; \
for (j=0; j<OBMAX; j++) { \
obDist = now.arrObs[j].d; \
obAng = now.arrObs[j].a; \
GET_OB_REL_TO_ROBOT(); \
if (inCone==1) {break;} \
} \
};
```

## 1.3. The Inline File

The file `newExMoInLines.txt` contains code for the inline function `HEAD_ON`, and definitions of the six environments, namely `SETENV1` to `SETENV6`.

```
inline HEAD_ON() {
/*Promela inline to add nondeterminism. Either we go right or we go left.*/
if
:: (headOn == 1) -> c_code{ Printf("HEADON1 \n");
x = 6;
now.sig = 6;
roboA = ((roboA+360)+ (x*OMEGAP) + CRASHANGFROMCENTRE);
roboA = ((int)(2*roboA)) - ((int)roboA);
roboA = ((int)roboA)%360; };
:: (headOn == 1) -> c_code{ Printf("HEADON2 \n");
x = (-6);
now.sig = (-6);
roboA = ((roboA+360) - (6*OMEGAP) - CRASHANGFROMCENTRE);
roboA = ((int)(2*roboA)) - ((int)roboA);
roboA = ((int)roboA)%360; };
:: (headOn != 1) -> c_code{Printf("HEADON3 BREAK \n");};
fi;
/*Now we assign the new value.*/
c_code{ now.roboAng = ((int)(2*roboA)) - ((int)roboA); };
headOn = 0;
};

inline SETENV1(){arrObs[0].d = 70; arrObs[0].a = 18;
arrObs[1].d = 122; arrObs[1].a = 274;
arrObs[2].d = 87; arrObs[2].a = 180
};

inline SETENV2(){arrObs[0].d = 70; arrObs[0].a = 18;
arrObs[1].d = 122; arrObs[1].a = 122;
arrObs[2].d = 87; arrObs[2].a = 216
};

inline SETENV3(){arrObs[0].d = 122; arrObs[0].a = 45;
arrObs[1].d = 121; arrObs[1].a = 326;
arrObs[2].d = 42; arrObs[2].a = 186
};

inline SETENV4(){arrObs[0].d = 90; arrObs[0].a = 0;
arrObs[1].d = 122; arrObs[1].a = 93;
arrObs[2].d = 68; arrObs[2].a = 199
};

inline SETENV5(){
arrObs[0].d = 60; arrObs[0].a = 15;
arrObs[1].d = 122; arrObs[1].a = 263;
arrObs[2].d = 100; arrObs[2].a = 175
}

inline SETENV6(){arrObs[0].d = 122; arrObs[0].a = 350;
arrObs[1].d = 121; arrObs[1].a = 69;
arrObs[2].d = 121; arrObs[2].a = 149;
arrObs[3].d = 121; arrObs[3].a = 229};
```

## 1.4. The Explicit Shell script

The shell script `verifyExplModel6.sh` contains instructions to compile and run the explicit model. Note that the upper memory limit is set using the `MEMLIM` compile time option, and the size of the search stack is set using the -m verification option. Definitions of all compile time and run time options can be found from the spin online reference pages at `http://spinroot.com/spin/Man/`.

```
spin -a ExplWithCcodeM6.pml;
gcc -DMEMLIM=2048 -O2 -DXUSAFE -w -o pan pan.c -lm;
./pan -m1000000 -a;
```

## 1.5. The Environment generation file

The file for generating a random environment is `explObGenV211.c`. To run this file use the following commands:

```
gcc explObGenV000.c -lm
./a.out
```

The number of environments generated, environment size, maximum number of obstacles and minimum distance between obstacles can be modified by adjusting the value of the corresponding constants. Full instructions are given in the code, but omitted from the description below. Note that to run this file you also need file `obGenFunctions.h`.

```
#include <math.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "obGenFunctions.h"

int main() {

FILE *fp;
fp = fopen("validEnviObCoords.txt", "w");

/**SET NUMBER OF ENVIRONMENTS**/
int NUMOFENVIS = 3;
/*****************************/

/**SET ENVIRONMENT SIZE**/
int ENVIRADIUS = 200;
/***********************/

/**SET MAXIMUM NUMBER OF OBSTACLES**/
int MAXOBNUM= 10;
/**********************************/

/**SET MINIMUM DISTANCE BETWEEN OBSTACLES**/
int MINDIST = 155;
/*****************************************/

/*Calc min dist from edge that ob can be
placed based on envi size and min dist between obs.*/
int MINFROMEDGE = ENVIRADIUS-((MINDIST/2)+1);
/*Random gen seed for init ob coord and
placement preference from obs (clock/anti).*/
srand(time(NULL));
int antiClock = rand() % 2;
/************************************************/

int envi = 0;
int i = 0;
int j = 0;
int k = 0;
int resetCount = 0;
int toClose = 0;
int fail = 0;
int test = 0;

int enviDist = 0;
int enviAng = 0;
int fromObAng = 10; /*Angle ob is to previous one*/
int angFrom = 0;
int obCount = 0;
int debug = 0;

coord_t nullifier;
nullifier.d = 0;
nullifier.a = 0;
coord_t prevOb;

fprintf(fp, "\nEnvironment generation code v2.11., author: Ryan Kirwan.\n\n");

fprintf(fp, "Environmental parameters: \n");
fprintf(fp, " Environment radius: %i.\n", ENVIRADIUS);
fprintf(fp, " Maximum number of obstacles requested: %i.\n", MAXOBNUM);
fprintf(fp, " Minimum distance between obstacles: %i.\n\n", MINDIST);

for (envi=1; envi<=NUMOFENVIS; envi++) {

/*Reset vars for new Envi.*/
i = 0; j = 0; k = 0; toClose = 0; fail = 0; obCount = 0;
enviDist = rand() % MINFROMEDGE;
/*There shall be no obs within 90 of the centre*/
if (enviDist <= 90) {enviDist = enviDist+90;}
enviAng = rand() % 360;
prevOb = setCoord(enviDist, enviAng);
arrObs[0] = setCoord(enviDist, enviAng);

fprintf(fp, "Enviroment E%i obstacles: \n", envi);
fprintf(fp, " arrObs[%d].d = %d; arrObs[%d].a = %d; \n", i, arrObs[0].d, i, arrObs[0].a);

for (i=1; i<=MAXOBNUM; i++) {

for (j=0; j<360; j++) {
toClose = 1;

if (antiClock==1) {angFrom = (359 - j);}
else {angFrom = j;}

arrObs[i] = genOb(prevOb, angFrom);

if (arrObs[i].d <= MINFROMEDGE) {
toClose = 0;

/*Loop through all preveous obstacles to check distance.*/
for (k=0; k<(i-1); k++) {

test = testDist(arrObs[i], arrObs[k]);

if (test < MINDIST) {
toClose = 1;
break;
}
}
}
if (toClose==0) {

antiClock = rand() % 2;
break;
}
if ((toClose==1) && (j==359)) {fail = 1;}
}

if (fail==1) {
      fprintf(fp, "\n Can't place more obstacles for this environment.\n\n"); break;}
else {obCount++;}

if (obCount >= MAXOBNUM) {
      fprintf(fp, "\n Successfully generated %i obstacles.\n\n", obCount); break;}

/*arrObs[0].d=80; arrObs[0].a=90;*/
  fprintf(fp, " arrObs[%d].d = %d;
  arrObs[%d].a = %d; \n",
                                  i, arrObs[i].d, i, arrObs[i].a);
/*Reset to the new coords of the latest obstacle.*/
  fromObAng = fromObAng + 90;
/*Save the previous Object coords for the obGen function.*/
  prevOb = setCoord(arrObs[i].d, arrObs[i].a);
}

}

fclose(fp);
return 0;
}
```

## 2. Abstract model

To verify the Abstract model you need the main Promela file, `AbsWithCcodeM12.pml`, and files `newAbsMo.h` and `inlineObGeneration.txt` containing C functions and inline functions respectively. To run verifications, use the shell script `verifyAbsModel12.sh`.

### 2.1. Main Promela file

This is the main Promela file, namely `AbsWithCcodeM12.pml`. To run properties P1: and P2: from the paper, select the appropriate ltl statement at the end of the main Promela file as before.

```
/*Abstract Model: Using Functions (Macros)*/

c_decl { #include <math.h> }
#include "newAbsMo.h"
#include "inlineObGeneration.txt"
#define MAX_OMEGAD 6

int obDist = 90;
int obAng = 0;
int omegaD = 0;
byte freeSpace = 1;
byte pLearn = 0;
int prevSig = 0;
int sig = 0;

/*Setting the C_Track variables*/
c_track "&x" "sizeof(double)"
c_track "&prevX" "sizeof(double)"
c_track "&obD" "sizeof(double)"
c_track "&obA" "sizeof(double)"
c_track "&dOmegaD" "sizeof(double)"

active proctype moving()
{
do
:: ((obDist > 30) && (freeSpace == 0)) ->
            d_step{ABS_RESPOND_TO_OB_BY_TURNING();}; d_step{ABS_MOVE_FORWARD();AB_LEARN();};
:: ((obDist <= 30) || (freeSpace == 1)) -> atomic{GENERATE_NEW_OB();};
od;
}


#define p ((sig<=-6)&&(sig>=6))
#define d ((prevSig!=0) && (prevSig>-6) && (prevSig<6) )
#define a (omegaD <= MAX_OMEGAD)



/*ltl p1 { <>[](p->!d)}*/
ltl p2 { [] a }
```

### 2.2. C function file

This is file `newAbsMo.h`. It contains C code defining constants and functions, as desribed in the paper. As before, the version below has all debugging code removed, see the actual code for this.

```
c_code {
#define PI 3.14159265
#define ROBORAD 20
#define OBRAD 10
#define AAL 11
#define AAR 69
#define ADIST 8
#define SENLEN 80
#define LAMBDA 1
```

```
#define OMEGAP 15
double obD = 0;
double obA = 0;
double theta = 0;
double opp = 0;
double adj = 0;
double sAdj = 0;
double hyp = 0;
double sHyp = 0;
double x = 0;
double prevX = 0;
double dOmegaD = 0;
double oAMAX = 0;
double lowR = 0;
double highR = 0;
double contactPoint = 0;
int i = 0;
signed int flag = 0;

int debug = 0;
};

#define AB_MOVE_FORWARD() c_code { \
obD = (double) now.obDist; \
obA = (double) now.obAng; \
theta = fabs(obA - 40);  \
adj = ( obD*cos(theta*(PI/180)) ); \
opp = sqrt( (obD*obD) - (adj*adj) ); \
adj = (adj - 1); \
obD = sqrt( (opp*opp) + (adj*adj) ); \
theta = (atan(opp/adj))*(180/PI); \
if (obA < 40) { obA = 40 - theta; } \
else { obA = 40 + theta; } \
if ((obA < 0) | (obA >= 80)){ obA = 0; obD = 90; now.freeSpace = 1;} \
now.obDist = ((int)(2*obD)) - ((int)obD); \
now.obAng = ((int)(2*obA)) - ((int)obA); \
};

#define AB_RESPOND_TO_OB_BY_TURNING() c_code { \
obD = (double)now.obDist; \
obA = (double)now.obAng; \
dOmegaD = (double)now.omegaD; \
oAMAX = (atan((OBRAD/obD)))*(180/PI); \
lowR = (obA - oAMAX); \
if (lowR < 0) { lowR = 0;} \
highR = (obA + oAMAX); \
if (highR > 80) { highR = 80;} \
if ( (AAL >= lowR) & (AAL <= highR) ) { theta = fabs(AAL - obA); flag = -1;} \
else if ( (AAR >= lowR) & (AAR <= highR) ) { theta = fabs(AAR - obA); flag = 1;} \
else {flag = 0;} \
opp = obD*(sin(theta*(PI/180)));\
if ((flag != 0 ) && (opp <= OBRAD)) { \
adj = sqrt((obD*obD) - (opp*opp)); \
if (adj <= 80) { \
for (i=0; i<=10; i++) { \
sAdj = OBRAD - i; \
sHyp = sqrt((sAdj*sAdj) + (opp*opp));  \
if ((sHyp != 0) & (sHyp <= OBRAD)) {  \
contactPoint = adj - sAdj; \
x = (flag)*(ADIST - ( ( contactPoint-(((int)contactPoint)%10) )/10 )); \
x = ((int)(2*x)) - ((int)x); \
break; \
} \
} \
} \
else { x = 0;} \
} \
else { x = 0;}  \
if (fabs(x) > 5) { \
obA = obA + (x*OMEGAP); \
now.pLearn = 1; \
} \
else { obA = obA + (x*dOmegaD);} \
if ((obA < 0) | (obA >= 80)) { now.freeSpace = 1;} \
else { now.obAng = ((int)(2*obA)) - ((int)obA);} \
now.prevSig = now.sig; \
now.sig = (int)(fabs(x)); \
};

#define AB_LEARN() c_code { \
if ((now.pLearn == 1) && (fabs(prevX) > 0) && (fabs(prevX) <= 5)) { \
now.omegaD = now.omegaD + LAMBDA; \
now.freeSpace = 1; \
} \
now.pLearn = 0; \
prevX = fabs(x); \
/*now.prevSig = (int)(fabs(prevX));*/ \
};
```

## 2.3. Object generation file

This file `inlineObGeneration.txt` contains the inline function to (nondeterministically) generate a new object at the edge of the environment when the robot is in free-space.

```
inline GENERATE_NEW_OB()  {

atomic {sig = 0; prevSig = 0;
obDist = 90; freeSpace = 0;
};

c_code {
x=0; prevX = 0;
};

do
:: atomic { obDist = 90; obAng=1;}; break;
:: atomic { obDist = 90; obAng=2;}; break;
:: atomic { obDist = 90; obAng=3;}; break;
:: atomic { obDist = 90; obAng=4;}; break;
:: atomic { obDist = 90; obAng=5;}; break;
:: atomic { obDist = 90; obAng=6;}; break;
:: atomic { obDist = 90; obAng=7;}; break;
:: atomic { obDist = 90; obAng=8;}; break;
:: atomic { obDist = 90; obAng=9;}; break;
:: atomic { obDist = 90; obAng=10;}; break;
:: atomic { obDist = 90; obAng=11;}; break;
:: atomic { obDist = 90; obAng=12;}; break;
:: atomic { obDist = 90; obAng=13;}; break;
:: atomic { obDist = 90; obAng=14;}; break;
:: atomic { obDist = 90; obAng=15;}; break;
:: atomic { obDist = 90; obAng=16;}; break;
:: atomic { obDist = 90; obAng=17;}; break;
:: atomic { obDist = 90; obAng=18;}; break;
:: atomic { obDist = 90; obAng=19;}; break;
:: atomic { obDist = 90; obAng=20;}; break;
:: atomic { obDist = 90; obAng=21;}; break;
:: atomic { obDist = 90; obAng=22;}; break;
:: atomic { obDist = 90; obAng=23;}; break;
:: atomic { obDist = 90; obAng=24;}; break;
:: atomic { obDist = 90; obAng=25;}; break;
:: atomic { obDist = 90; obAng=26;}; break;
:: atomic { obDist = 90; obAng=27;}; break;
:: atomic { obDist = 90; obAng=28;}; break;
:: atomic { obDist = 90; obAng=29;}; break;
:: atomic { obDist = 90; obAng=30;}; break;
:: atomic { obDist = 90; obAng=31;}; break;
:: atomic { obDist = 90; obAng=32;}; break;
:: atomic { obDist = 90; obAng=33;}; break;
:: atomic { obDist = 90; obAng=34;}; break;
:: atomic { obDist = 90; obAng=35;}; break;
:: atomic { obDist = 90; obAng=36;}; break;
:: atomic { obDist = 90; obAng=37;}; break;
:: atomic { obDist = 90; obAng=38;}; break;
:: atomic { obDist = 90; obAng=39;}; break;
:: atomic { obDist = 90; obAng=40;}; break;
:: atomic { obDist = 90; obAng=41;}; break;
:: atomic { obDist = 90; obAng=42;}; break;
:: atomic { obDist = 90; obAng=43;}; break;
:: atomic { obDist = 90; obAng=44;}; break;
:: atomic { obDist = 90; obAng=45;}; break;
:: atomic { obDist = 90; obAng=46;}; break;
:: atomic { obDist = 90; obAng=47;}; break;
:: atomic { obDist = 90; obAng=48;}; break;
:: atomic { obDist = 90; obAng=49;}; break;
:: atomic { obDist = 90; obAng=50;}; break;
:: atomic { obDist = 90; obAng=51;}; break;
:: atomic { obDist = 90; obAng=52;}; break;
:: atomic { obDist = 90; obAng=53;}; break;
:: atomic { obDist = 90; obAng=54;}; break;
:: atomic { obDist = 90; obAng=55;}; break;
:: atomic { obDist = 90; obAng=56;}; break;
:: atomic { obDist = 90; obAng=57;}; break;
:: atomic { obDist = 90; obAng=58;}; break;
:: atomic { obDist = 90; obAng=59;}; break;
:: atomic { obDist = 90; obAng=60;}; break;
:: atomic { obDist = 90; obAng=61;}; break;
:: atomic { obDist = 90; obAng=62;}; break;
:: atomic { obDist = 90; obAng=63;}; break;
:: atomic { obDist = 90; obAng=64;}; break;
:: atomic { obDist = 90; obAng=65;}; break;
:: atomic { obDist = 90; obAng=66;}; break;
:: atomic { obDist = 90; obAng=67;}; break;
:: atomic { obDist = 90; obAng=68;}; break;
:: atomic { obDist = 90; obAng=69;}; break;
:: atomic { obDist = 90; obAng=70;}; break;
:: atomic { obDist = 90; obAng=71;}; break;
:: atomic { obDist = 90; obAng=72;}; break;
:: atomic { obDist = 90; obAng=73;}; break;
:: atomic { obDist = 90; obAng=74;}; break;
:: atomic { obDist = 90; obAng=75;}; break;
:: atomic { obDist = 90; obAng=76;}; break;
:: atomic { obDist = 90; obAng=77;}; break;
:: atomic { obDist = 90; obAng=78;}; break;
:: atomic { obDist = 90; obAng=79;}; break;
:: atomic { obDist = 90; obAng=80;}; break;
od;
};
```

## 2.4. The Abstract Shell script

The shell script `verifyAbsModel12.sh` contains instructions to compile and run the abstract model. Options are set as for the Explicit model.

```
spin -a AbsWithCcodeM12.pml;
gcc -DMEMLIM=2048 -O2 -DXUSAFE -w -o pan pan.c -lm;
./pan -m1000000 -a;
```