A tool for modelling, verifying, debugging sC++ programs using Promela/Spin/Xspin

Thierry CATTEL[1], Gregory DUVAL[1]

[1] Computer Networking Laboratory, Swiss Federal Institute
CH-1015 Lausanne, Switzerland
{Thierry.Cattel, Gregory.Duval}@epfl.ch

Abstract. We demonstrate a tool useful to model, verify, and debug concurrent applications written in sC++. The tool is written in Yacc and C++. It provides support for the systematic translation of sC++ programs into Promela, the verification of the resulting models through SPIN, the simulation and debugging of the sC++ program using XSPIN.

1 sC++

sC++[1] is an extension of C++ that adds concurrency to the language by unifying the object and task concepts into a single one, the active object. The management of active objects is the same as the management of usual C++ passive objects. The difference is that active objects have their own behaviour and that they may either accept method calls or call other objects'methods. They are also capable to await simultaneously several non deterministic events including timers expiration. sC++ is supported by a clone of *gcc* for compilation and code generation. It is based on a micro-kernel relying on coroutines.

2 Tool Implementation

The tool (Fig.1) is composed of a translator *scxx2promela*, written in Yacc and C++, that translates sC++ source into Promela source. It also generates a correspondence table that implements a function that relates the line number of the produced code to the line number of the source code. This table is used by a modified version of XSPIN. This allows to do simulations and debugging directly on the sC++ code. When loaded a sC++ program is translated in two files. A first one contains the Promela code and this is the actual file that is used to perform the simulation and verification. The second one contains the table which is used each time a display operation is done in the simulator. The all approach and the formalization of the translator is presented in [3].

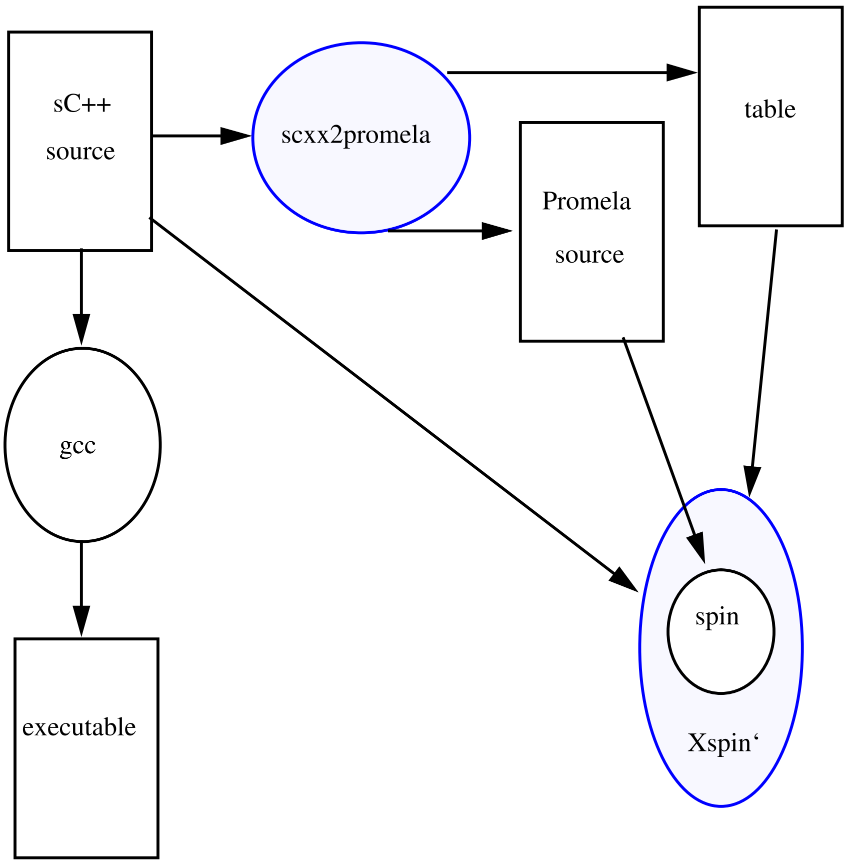

FIGURE 1. Tool architecture

3 References

1. C. Petitpierre, sC++, A Language Adapted to Interactive Applications, accepted for IEEE Computer Journal, March 1998. http://diwww.epfl.ch/w3lti/
2. G.J. Holzmann, The Model Checker SPIN, IEEE Transactions on software engineering, vol. 23, no.5, May 1997.
3. T. Cattel, Modelling and verfication of sC++ applications, TACAS‘98, Lisbon, Aprl 1998.

4 Example of sC++ program

```
#include <stdio.h>
#include <scxx.h>
#include <assert.h>

int in;

active class Semaphore{
  int n;
  @Semaphore(){
    n = 2;
    while(true){
      select{
        when(n>0)
          accept P;
      ||
        accept V;
      };
    };
  }
 public:
  void P(){ n--;}
  void V(){ n++;}
};
active class User{
  int pid;
  Semaphore* s;
  @User(){
    accept minit;
    while(true){
      s->P();
      in++;
      printf(“User %d in CS\n”,pid);assert(in<=2);
      in--;
      s->V();
    };
  }
 public:
  void minit(Semaphore *S,int p){ s = S;pid = p;}
};
void main(){
  Semaphore* s=new Semaphore();
  User* u1=new User();
  User* u2=new User();
  u1->minit(s,1);
  u2->minit(s,2);
  delete s;
}
```

5 Translation in Promela

```
#define NULL 255
#define NBMAXINST 5

#define Semaphore_ byte
#define User_ byte

int in;

typedef Semaphore{
  chan P_c[3]=[0] of {bit}
  chan P_r=[0] of {bit}
  chan V_c[3]=[0] of {bit}
  chan V_r=[0] of {bit}
  chan dest_c[3]=[0] of {bit}
  chan dest_r=[0] of {bit}
}
Semaphore Semaphore_inst[NBMAXINST];
byte Semaphore_instnb=0;

typedef User{
  chan minit_c[3]=[0] of {bit,Semaphore_,int}
  chan minit_r=[0] of {bit}
  chan dest_c[3]=[0] of {bit}
  chan dest_r=[0] of {bit}
}
User User_inst[NBMAXINST];
byte User_instnb=0;

proctype Semaphore_proc(byte this){
  int _pi;
  int n;
  n = 2;
  do
  :: Semaphore_inst[this].P_c[2-(n>0)]?true;
       n--;
     Semaphore_inst[this].P_r!true;
  :: Semaphore_inst[this].V_c[2-(true)]?true;
       n++;
     Semaphore_inst[this].V_r!true;
  od;
  Semaphore_inst[this].dest_c[2-(true)]?true;
  Semaphore_inst[this].dest_r!true;
  goto end_Semaphore;
end_Semaphore:skip
}
```

```
proctype User_proc(byte this){
  int _pi;
  int pid;
  Semaphore_ s;
  Semaphore_ S;
  int p;
  User_inst[this].minit_c[2-(true)]?true, S, p;
    s = S;
    pid = p;
  User_inst[this].minit_r!true;
  do
  :: Semaphore_inst[s].P_c[(true)]!true;
     Semaphore_inst[s].P_r?true;
     in++;
     printf(“User %d in CS\n”,pid);
     assert(in<=2);
     in--;
     Semaphore_inst[s].V_c[(true)]!true;
     Semaphore_inst[s].V_r?true;
  od;
  User_inst[this].dest_c[2-(true)]?true;
  User_inst[this].dest_r!true;
  goto end_User;
end_User:skip}

init{
  int _pi;
  Semaphore_ s;
  User_ u1;
  User_ u2;
  atomic{
    s=Semaphore_instnb;
    run Semaphore_proc(s);
    Semaphore_instnb++;}
  atomic{
    u1=User_instnb;
    run User_proc(u1);
    User_instnb++;}
  atomic{
    u2=User_instnb;
    run User_proc(u2);
    User_instnb++;}
  User_inst[u1].minit_c[(true)]!true,s,1;
  User_inst[u1].minit_r?true;
  User_inst[u2].minit_c[(true)]!true,s,2;
  User_inst[u2].minit_r?true;
  Semaphore_inst[s].dest_c[true]!true;
  Semaphore_inst[s].dest_r?true;
}
```