verds: Verification of Hierarchical Discrete Systems by Symbolic Techniques

Wenhui Zhang
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
P.O.Box 8718, Beijing 100190, China

1 Introduction

verds[1] is a tool for the verification of hierarchical discrete systems by symbolic methods. This tool has integrated a TBD-based model checking approach [7] (a kind of symbolic model checking [5, 3]) and two bounded correctness checking approaches (a kind of bounded model checking [2, 1] based on bounded semantics of CTL [6]). The model is a kind of fair discrete systems, which may be specified hierarchically with a recursively defined semantics.

2 Modeling and Specification

The modeling language is called VERDS modeling langauge (VML), and the property specification langauge is CTL [4]. The reader is referred to [8] for a description of VML. A verification model specified in VML is called a VERDS verification model (VVM).

Checking C-Programs There is a tool for transforming (a subset of) C-programs into VVMs. Then the C-programs may be verified by *verds*. Currently the specification of the properties of C-programs is of the form

(at line n): bexp

where n is a line number of the program such that the number points to a line in the *main*-function, and *bexp* is a boolean expression. The meaning of the specification is that whenever the program execution reaches the line, the assertion *bexp* must hold.

3 Examples

This section contains an example of checking a mutual exclusion algorithm and an example of checking a C-program.

[1] http://lcs.ios.ac.cn/∼zwh/verds/

3.1 Checking a Mutual Exclusion Algorithm

The following (Fig. 1) is a specification of a mutual exclusion algorithm with two processes in VML, and with the following properties: mutual exclusion, progress, non-starvation, cooperative non-starvation.

```
VVM    me005
VAR    x[0..1]:0..1; t:0..1;
INIT   x[0]=0; x[1]=0; t=0;
PROC   p0:p0m(x[],t,0); p1:p0m(x[],t,1);
SPEC
       AG(!(p0.a=s2&p1.a=s2));
       AG((!p0.a=s1|AF(p0.a=s2|p1.b=s2))&(!p1.b=s1|AF(p0.a=s2|p1.b=s2)));
       AG((!p0.a=s1|AF(p0.a=s2))&(!p1.b=s1|AF(p1.b=s2)));
       AG((!p0.a=s1|EF(p0.a=s2))&(!p1.b=s2|EF(p1.b=s2)));

MODULE p0m(x[],t,i)
VAR    a: {s0,s1,s2,s3};
INIT   a=s0;
TRANS
       a=s0:                (x[1-i],t,a):=(1,1-i,s1);
       a=s1&(x[i]=0|t=i):   (a):=(s2);
       a=s1&!(x[i]=0|t=i):  (a):=(s1);
       a=s2:                (x[1-i],a):=(0,s3);
       a=s2:                (a):=(s2);
       a=s3:                (x[1-i],t,a):=(1,1-i,s1);
FAIRNESS running;
```

Fig. 1. The Example Mutual Exclusion Algorithm

TBD-based Model Checking Suppose that this verification model is contained in the file *me*005.*vvm*. For verification of the two properties with TBD-based model checking, we use the following command:

```
verds -ck i me005.vvm
```

in which i must be instantiated to one of $\{1, 2, 3, 4\}$ for checking the specified properties. The verification result shows that the first two properties and the last property are true and the third property is false. The third property does not hold, because a process may stay at the critical region while the other process keep testing whether the condition for entering the critical region holds. The property holds when the fairness specification $a! = s2$ (meaning that a process cannot constantly stay at the critical region) is added to both of the processes.

Bounded Correctness Checking Verification by bounded correctness checking can currently handle CTL formulas and models without fairness. Let the file me004.vvm be the modification of me005.vvm such that the fairness constraint is removed. For verification of the two properties using bounded semantics (without fairness), we use the following command:

```
verds -bs -ck i me004.vvm
```

The verification result shows that the first property and the last property are true while the second property and the third property are false without the fairness constraints. In addition, it is reported that the bounds are respectively 10, 2, 2 and 10 for verification and falsification of the properties.

3.2 Checking a C-Program

The following (Fig. 2) is a specification of a program that read a number between 0 and 9, call the function $f91()$ with this number as the argument, and print the result returned from this function.

```
#include <stdio.h>
/***********************************************/
main(int argc,char **argv ) {
  int n=0,m=0;
  printf("INFO:   system is now active\n",0);
  while (1) {
    n=in(); m=f91(n);
    printf("RESULT: %i\n\n",m);
  }
}
/***********************************************/
int in() {
  char c=0;
  char d=0;
  while (1) {
    putc('N',stdout); putc(':',stdout); putc(9,stdout); c=getc(stdin);
    if (c<'0'||c>'9') {
      while (1) { c=getc(stdin); if (c=='\n') break; }
      printf("INFO:   the input must be a digit\n\n"); continue;
    }
    d=getc(stdin); if (d=='\n') { return c-'0'; }
    printf("INFO:   the input must be 1 digit\n\n");
  }
}
/***********************************************/
int f91(int x) {
  int y1=x; int y2=1; int z=0;
  while (1) {
    if (y1>100) {if (y2!=1) { y1=y1-10; y2=y2-1; } else { z=y1-10; break; }}
    else { y1=y1+11; y2=y2+1; }
  }
  return z;
}
/***********************************************/
```

Fig. 2. The Example C-Program

Model Checking Let the property of the program be specified as follows

```
(at line 9): m==91
```

Assume that this specification is contained in the file $f091.sp$ and the program is contained in the file $f091.c$. For verification of whether the program is correct against this specification, we use the following command:

```
verds -c f091.c -sp f091.sp
```

The verification result shows that the property holds.

4 Compositional Reasoning

Compositional reasoning may be applied to replace a function with what the function call guarantees. Take the C-program as an example. Let the assumption-guarantee pair of the function $in()$ be specified by

```
FUNCTION    r=in()
ASSUMPTION  TRUE;
GUARANTEE   0<=r&&r<=9;
```

Assume that this specification is contained in the file $f091.fsp$. For the verification of whether the program is correct against the specification in $f091.sp$ with this assumption-guarantee specification, we use the following command:

```
verds -c f091.c -sp f091.sp -fsp f091.fsp
```

The verification result shows that the property holds with the assumption-guarantee specification, and with improved efficiency of the verification. This command also produces the file $f091.vvm$ which contains the model (VERDS verification model) used in the verification. Then we may verify whether the function $in()$ satisfies the assumption-guarantee specification with the following command:

```
verds -ck in f091.vvm
```

The verification result shows that this also holds.

Remarks For this simple example, the compositional reasoning approach may not have advantage (considering the total time of verifying both the assumption-guarantee specification and the C-program with this specification), since there is a lot overhead involved in the application of the compositional reasoning. Generally, it is useful and increases the scalability of the model checking approach.

5 Concluding Remarks

The verification tool *verds* with its modeling langauge and specification language has been presented. A simple example has also been presented to show the use of the languages and the verification tool. Two distinguished features of *verds* are that it implements TBD-based model checking and QBF-based bounded correctness checking, and for some applications, it complies with compositional reasoning.

References

1. A. Biere, A. Cimmatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded Model Checking. Advances in Computers 58, Academic Press, 2003.
2. A. Biere, A. Cimmatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. LNCS 1579:193-207. TACAS 99.
3. E. M. Clarke, O. Grumberg and D. Peled. Model Checking. The MIT Press. 1999.
4. E. Allen Emerson and E. M. Clarke. Using Branching-time Temporal Logics to Synthesize Synchronization Skeletons. *Science of Computer Programming* 2(3): 241-266. 1982.
5. K. L. McMillan. Symbolic Model Checking. Kluwer Academic Publisher,1993.
6. W. Zhang. Bounded Semantics of CTL. Technical Report ISCAS-LCS-16, Institute of Software, Chinese Academy of Sciences. 2010.
7. W. Zhang. Ternary Boolean Diagrams. Technical Report ISCAS-LCS-10-24, Institute of Software, Chinese Academy of Sciences. 2010.
8. W. Zhang. VERDS Modeling Language. Manuscript, SKLCS, Institute of Software, Chinese Academy of Sciences. 2012.