DMC user guide

Giorgio Delzanno
DISI - University of Genova
In collaboration with Andreas Podelski
MPI - Max-Planck-Institut für Informatik, Saarbruecken
email: giorgio@disi.unige.it

January 15, 2002

Contents

1 What is DMC?

DMC (Deductive Model Checking) is a prototype symbolic model checker implemented in SICSsus Prolog for infinite-stte systems with integer and real data variables. The symbolic representation of the state-space is given through a collection (disjunction) of *linear arithmetic constraints*. Cosntraints are stored in the internal database of SICStus Prolog as *constrained facts* (i.e. unit facts with constraints in the body). Constraints are handled by the internal real-solver of SICStus (Holzbaur's clp(Q,R) library). Safety and liveness properties are verified using a possibly non terminating (nested) fixpoint computation. A set of acceleration rules (defined in the rule-based style typical of Prolog) are used to speed-up (and sometimes) enforce termination of the computation. With DMC, we managed to prove properties of concurrent systems like mutual exclusion algorithms (Bakery, Ticket, Producer-Consumers), parameterized cache coherence protocols (Illinois, Berkeley, Dragon, Firefly) and abstractions of C-programs for array bounds checking. We describe next how to load and experiment using the current prototype version and the cache coherence protocols in the directory 'examples'.

References

[Del00] G. Delzanno. Automated Verification of Parameterized Cache Coherence Protocols. Technical Report DISI-TR-00-1, DISI, University of Genova, January 2000. Available at the address http://www.disi.unige.it/person/DelzannoG/papers.

[DEP99] G. Delzanno, J. Esparza, and A. Podelski. Constraint-based Analysis of Broadcast Protocols. In *Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL'99)*, volume 1683 of LNCS, pag. 50–66. Springer, 1999.

[DP99] G. Delzanno and A. Podelski, Model Checking in CLP. In W. R. Cleaveland, editor, *Proc. 5th Int. Con. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99).*, LNCS 1579, pages 223–239. Springer-Verlag, 1999.

[DP99b] G. Delzanno and A. Podelski, Constraint-based Deductive Model Checking in CLP. Invited to a Special Issue of *Software Tools for Technology Transfer*. Available at the address http://www.disi.unige.it/person/DelzannoG/papers.

2 The Command-line Version

In this section we explain how to use DMC from a Prolog interpreter command line. This explanation is instantiated to the application on cache coherence protocol verification (i.e. here we don't discuss liveness properties).

2.1 How to invoke DMC

- Move to the directory '../dmc/' with the Prolog code of dmc (demo.pl, checker.pl, etc).
- Invoke 'sicstus'
- You are now in the interactive Prolog interpreter where you can submit goal with the syntax '?-Goal.' (Goal must be a Prolog atom).
- Load 'demo.pl' using the command '?-[demo]'.
- Now you are able to use the DMC commands.

2.2 DMC commands

- **ld(filename)** to load the description of a system from the directory '.../dmc/examples' (i.e. please save your examples there).
- **lr** to see the loaded system.
- **ls** to see the current set of computed (reachable) states.
- **prop(P,S)** too see the input property to check.
- **vn** to verify a property (specified in the input system) without accelerations. E.g., for safety properties **vn** computes (when terminating) all backward reachable states.
- **vt** to verify a property (specified in the input system) without accelerations with a test for the initial state.
- **va** to verify a property (specified in the input system) with accelerations (may take more time than vn).
- **ti** to check for the presence of the initial state in the resulting set of states.
- **tr** to compute an error trace.
- **ptime(Goal)** to compute the execution time for Goal.

- **clearS** to clean the database before new execution of one of **va**, **vn**, **vt**.

Warning: before executing one of **vn**, **va**, **vt** be sure that the internal database is clean (i.e. **ls.** must return no constrained facts, only **yes**). You can use **cleanS** to clean the internal database. A new **ld(file)** will automatically clean the database.

2.3 A sample session

(1) We first loda the example*mesi* from teh directory 'examples'; (2) we display the input system and the input property (in this case a *safety* property); (3) we run the checker asking for the execution time using **?-ptime(vn).**; (4) we check for the initial state to see whether the property is satisfied or not using **ti**. (5) we display the resulting backward reachable states using **ls**. (6) we clean the database using **cleanS** and go on.

```
| ?- ld(mesi).
{consulting /share/home/giorgio/demo/tmp/input_clp...}
{/share/home/giorgio/demo/tmp/input_clp consulted, 50 msec 2312 bytes}

yes
| ?- lr.

r(init, p(s_s,A,B,C,D), {A>=0,B=0,C=0,D=0}, 1).
r(p(s_s,A,B,C,D), p(s_s,A,B,C,D), {A>=0,B>=0,D>=0,C>=0,B+D+C>=1}, 2).
r(p(s_s,A,B,C,D), p(s_s,E,F,G,H), {A>=1,B>=0,C>=0,D>=0,E=A-1,F=B+D+C+1,G=0,H=0}, 3).
r(p(s_s,A,B,C,D), p(s_s,A,B,C,D), {D>=1,A>=0,C>=0,B>=0}, 4).
r(p(s_s,A,B,C,D), p(s_s,A,B,E,F), {A>=0,B>=0,D>=0,C>=1,E=C-1,F=D+1}, 5).
r(p(s_s,A,B,C,D), p(s_s,E,F,G,H), {A+B>=1,A>=0,B>=0,C>=0,D>=0,E=A+B+C+D-1,F=0,G=1,H=0}, 6).
r(p(s_s,A,B,C,D), p(s_s,E,B,F,D), {A>=0,B>=0,D>=0,C>=1,F=C-1,E=A+1}, 7).
r(p(s_s,A,B,C,D), p(s_s,E,F,C,D), {A>=0,B>=1,D>=0,C>=0,F=B-1,E=A+1}, 8).
r(p(s_s,A,B,C,D), p(s_s,E,B,C,F), {A>=0,B>=0,D>=1,C>=0,F=D-1,E=A+1}, 9).

yes
| ?- prop(P,S).

P = unsafe,
S = p:s*(s>=1)*(m>=1) ?        [here use: Enter]

yes
?- ptime(vn).

Execution time 0.5
yes
| ?- ti.

Property satisfied
yes
| ?- ls.

s(0, p(s_s,_,A,_,B), {B>=1,A>=1}, 1, (0,0)).
s(1, p(s_s,A,B,C,D), {C>=1.0,D>=0.0,B>=1.0,A>=0.0}, 2, (5,1)).

yes

| ?- clearS.

...
```

2.4 What to do in case DMC seems to diverge

If the command you submitted remains suspiciously suspended for some time (i.e. no answer (yes/no) is returned) then **vn, va, vt** will probably go on forever. You can interrupt DMC using Ctr-C and return to the interpreter level. If you used**vn** you can try again using **va** (that uses accelerations) as in the following example.

```
| ?- ld(firefly).
{consulting /share/home/giorgio/demo/tmp/input_clp...}
{/share/home/giorgio/demo/tmp/input_clp consulted, 50 msec 2312 bytes}

yes
| ?- prop(M,S).

M = p:s*(s>=0)*(i>=0)*(d>=2)*(e>=0),
S = unsafe ?                         [here: Enter]

yes
| ?- vn.                             [it seems not terminating: Ctrl-C]

Prolog interruption (h for help)?   [here: Enter]

| ?- clearS.

yes

| ?- va.

Widening on point-sequence
Widening on point-sequence
Widening on point-sequence
yes

| ?- ls.

s(0, p(s_s,A,B,C,D), {D>=0,B>=2,A>=0,C>=0}, 1, (0,0)).
s(1, p(s_s,A,B,C,D), {D>=1.0,B>=1.0,A>=0.0,C>=0.0}, 2, (7,1)).
s(2, p(s_s,A,B,C,D), {B>=0.0,D>=2.0,A>=0.0,C>=0.0}, 3, (7,2)).
s(2, p(s_s,A,B,1.0,C), {C>=0.0,B>=1.0,A>=0.0}, 4, (8,2)).
s(3, p(s_s,A,B,1.0,C), {C>=1.0,B>=0.0,A>=0.0}, 5, (7,4)).
s(3, p(s_s,A,B,2.0,C), {A>=0.0,C>=0.0,B>=1.0}, 6, (14,4)).
s(4, p(s_s,A,B,C,D), {(D>=0.0,B>=1.0,A>=0.0),C>=1.0}, 7, (w(7),0)).
s(4, p(s_s,A,B,C,D), {D=1.0-C,C=<1.0,C>= -0.0,A>=1.0,B>=1.0}, 8, (5,6)).
s(4, p(s_s,A,B,C,D), {(D>=0.0,B>=1.0,A>=0.0),C>=1.0}, 9, (w(9),0)).
s(4, p(s_s,A,B,2.0,C), {C>=1.0,B>=0.0,A>=0.0}, 10, (7,6)).
s(4, p(s_s,A,B,C,D), {(D>=0.0,B>=1.0,A>=0.0),C>=1.0}, 11, (w(11),0)).
s(5, p(s_s,A,B,C,D), {C>=0.0,D>=0.0,A>=1.0,B>=1.0,C+D>=1.0}, 12, (5,7)).
s(5, p(s_s,A,B,C,D), {D>=1.0,B>=0.0,A>=0.0,C>=1.0}, 13, (7,7)).
s(5, p(s_s,A,B,C,D), {C=2.0-D,D=<2.0,D>=1.0,B>=0.0,A>=1.0}, 14, (7,8)).
s(6, p(s_s,A,B,C,D), {D>=1.0,B>=0.0,A>=1.0,C>=0.0,D+C>=2.0}, 15, (7,12)).

| ?- ti.

Property satisfied
yes
```

3 Syntax of input file

The input file has two parts: the system and the property.

3.1 System

The input file must be in the directory examples. It is a Prolog input file (i.e. consisting of a collection of facts and clauses). Please us a name without extension (i.e., 'mesi', 'illinois', etc). We give immediately an example.

```
:-dynamic r/4.
:-dynamic info/4.

info(1,[p1],2,[a,b]).

r(p(s_think,X,Y),p(s_wait,X1,Y),{X1:=X+1},1).
r(p(s_wait,X,Y),p(s_use,X1,Y),{X1:=X+1},1).
```

The 'dynamic' declaration is always needed. The **info** predicate is used to associated symbolic names to system and data variables (otherwise treated as logical variables that have no real name). In the example there is one system variable (or state, or location) and 2 data variables (a,b).

Warning: If you want to write a new example remember that in the CLP input file the constants associated to the locations (**add a 'dummy' one if none**) must always be **prefixed** by s_ (like s_think, s_wait and **NOT** wait, think).

The transition rules are expressed as predicates $r(Head, Body, Constraint, Index)$ with 4 arguments. $Head$ and $Body$ are tuples of variables and constants (expressed using the constructor p) expressing the old and the new state respectively. Variables must ber written using names beginning wirth **upper-case** letters. The *Constraint* describes how the new and old state are related. *Constraint* is a SICStus prolog linear arithmetic constraint with syntax:

```
C ::= {} | {A,A,...,A}
A ::= E >= E | E =< E | E = E | E > E | E < E
E ::= E + E | E - E | c * E | Var | Const
Var ::= X,Y,X1,Y1, etc
Const ::= 0 | 1 | 0.0 | 1.0 | ....
```

3.2 Property

Properies are expressed using the predicate **prop(type,Property).** For the examples we are interested in, **type** must be *unsafe* (i.e. we are considering a safety property).

Property is an expression as follows. The name of variables are that of the field 'info' in the CLP/internal repr. ot that of the original specification. E.g. for our example info(1,[p1],2,[a,b]) means that the program locations for processes 1 is 'p1' and that the two data variables are 'a' and 'b'. Then you build **Property** as an expression defined as follows.

```
Atom::= Loc:Locvalue | ( Constraint )
Exp ::= Exp * Exp | Exp + Exp | Atom

Locvalue is a Prolog term (corresponding to the name
of the locations (eg. think,wait,use)).
```

```
Constraint is a Prolog constraint defined over the *NAMES*
of the data variables (e.g. a>=2, a =< 2, a = b, etc).
```

For instance, the following expression are well defined: p1:use + p1:use * (a >= b) + p1:use * (a =< 1), etc.

Remember to put the parenthesis around the constraints and to use the variables names and not Prolog variables (i.e. **NO**: P1:use * (A >= B)).

Warning: system and properties are stored together in the same file. As an example.

```
:-dynamic r/4.
:-dynamic info/4.

info(1,[p1],2,[a,b]).

r(p(s_think,X,Y),p(s_wait,X1,Y),{X1:=X+1},1).
r(p(s_wait,X,Y),p(s_use,X1,Y),{X1:=X+1},1).
prop(unsafe,p1:use * (a >=0) * (b>=0)).
```

Note: When writing properties you can forget about the prefix s_ for the locations and write: p1:use * (a >= 3) etc

3.3 The MESI protocol specification

The DMC description of the MESI protocol is given as follows.

```
:-dynamic r/4.
:-dynamic info/4.

info(1,[p],4,[i,s,e,m]).

r(init,p(s_s,I,S,E,M),{I>=0,S=0,E=0,M=0},1).

r(p(s_s,I,S,E,M),p(s_s,I,S,E,M),
    {I>=0,S>=0,M>=0,E>=0,S+M+E>=1},2).

r(p(s_s,I,S,E,M),p(s_s,I1,S1,E1,M1),
   {I>=1, S>=0,E>=0,M>=0,I1=I-1,
    S1=S+M+E+1,E1=0,M1=0},3).

r(p(s_s,I,S,E,M),p(s_s,I,S,E,M),
    {M>=1,I>=0,E>=0,S>=0},4).

r(p(s_s,I,S,E,M),p(s_s,I,S,E1,M1),
    {I>=0,S>=0,M>=0,E>=1,E1=E-1,M1=M+1},5).

r(p(s_s,I,S,E,M),p(s_s,I1,S1,E1,M1),
   {I+S>=1, I>=0,S>=0,E>=0,M>=0,
        I1=I+S+E+M-1,S1=0,E1=1,M1=0},6).

r(p(s_s,I,S,E,M),p(s_s,I1,S,E1,M),
    {I>=0,S>=0,M>=0,E>=1,E1=E-1,I1=I+1},7).
```

```
r(p(s_s,I,S,E,M),p(s_s,I1,S1,E,M),
    {I>=0,S>=1,M>=0,E>=0,S1=S-1,I1=I+1},8).

r(p(s_s,I,S,E,M),p(s_s,I1,S,E,M1),
    {I>=0,S>=0,M>=1,E>=0,M1=M-1,I1=I+1},9).

prop(unsafe,p:s * (s>=1) * (m>=1)).
%prop(unsafe,p:s * (m>=2)).
```

% is used for commenting lines.