Delta Debugging – Programming Exercises

SS 2007

Johannes Kepler University

Linz, Austria

Dr. Toni Jussila
Institut for Formal Models and Verification

http://fmv.jku.at/kv

(Nu)SMV

NuSMV is a widely applied model checking tool for finite state machines.

- both synchronous and asynchronous models
- supports modular design
- BDD based and bounded model checking (CTL and LTL)
- well established
  - 3rd party (companies & academic institutions) tools to read SMV models
- homepage `http://nusmv.irst.itc.it/`

Input Language

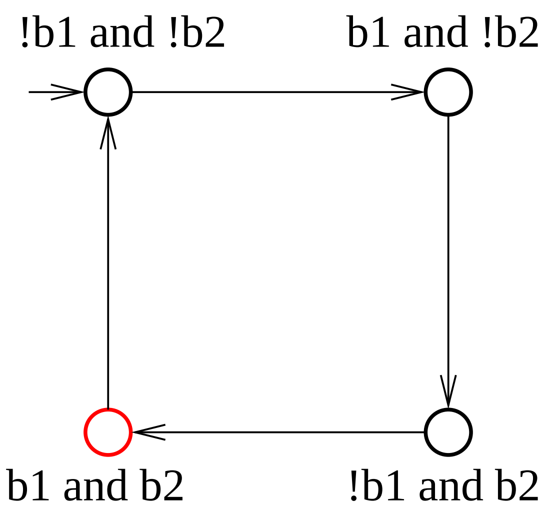
We will concentrate on **flat** (no modules) SMV modules with reachability properties and only Boolean variables.

Skeletal delta debugging algorithm:

1. eliminate $\mathcal{C}_x$ variables,
   - remove state variables (section `VAR`) and
   - replace references to removed variables with Boolean constants. Simplify. (sections `ASSIGN` and `DEFINE`)
2. if failure persists, update $\mathcal{C}_x$, and continue from 1.
3. try removing $\mathcal{C} \setminus \mathcal{C}_x$, if OK, update $\mathcal{C}_x$, and continue from 1,
4. otherwise, increase granularity (decrease $\mathcal{C}_x$).

Example (two-bit counter)

```
MODULE main
VAR
  b1: boolean;
  b2: boolean;
ASSIGN
  init(b1) := 0;
  init(b2) := 0;
  next(b1) := !b1;
  next(b2) := b1 xor b2;
DEFINE
  bad := b1 & b2;
SPEC AG !bad
```



Example: remove `b1`

Lines beginning with -- are comments.

```
MODULE main
VAR
  -- b1: boolean;
  b2: boolean;
ASSIGN
  -- init(b1) := 0;
  init(b2) := 0;
  -- next(b1) := !b1;
  next(b2) := 1 xor b2; -- can be simplified !b2;
DEFINE
  bad := 1 & b2; -- can be simplified to b2
SPEC AG !bad
```

Example: continue with `b2`

```
MODULE main
VAR
  -- b1: boolean;
  -- b2: boolean;
ASSIGN
  -- init(b1) := 0;
  -- init(b2) := 0;
  -- next(b1) := !b1;
  -- next(b2) := 1 xor b2;
DEFINE
  bad := 1 & 1; -- can be simplified to 1
SPEC AG !bad
```

Remarks

- Non-trivial SMV knowledge is required.
  - blindly commenting lines leads to syntactically incorrect models

- 2nd dimension: should variables be replaced by 0 or 1 or a combination thereof ?

- Boolean simplification is a potential source of error.

```
ib := (((gna | (!gna & ws)) & !(gna & ((!aya & ((vv & ((fj <-> 0)
0) & ((mi <-> 0) & ((iha <-> 1) & (li <-> 0))))))))))));
```

- Goal: to produce the *simplest* possible SMV model, therefore

- your output should not contain comments nor unsimplified Boolean expressions.

Tools to Test Against

- `NuSMV`
  - given a model, what is the smallest such model with the same truth value
  - real `NuSMV` bugs ??

- `smv2qbf`
  - translates SMV models to QBFs (encoding model checking problems)
  - contains real bugs
  - available from `http://fmv.jku.at/smv2qbf/`

Quantified Boolean Formulas

- propositional formula having a quantifier (∃/∀) prefix.

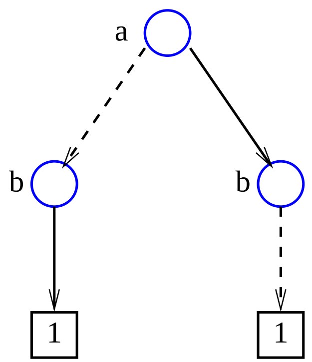
1. $(a \vee b) \wedge (\neg a \vee \neg b)$	$\Rightarrow$	SAT
2. $\forall a \exists b((a \vee b) \wedge (\neg a \vee \neg b))$	$\Rightarrow$	TRUE
3. $\forall b \exists a((a \vee b) \wedge (\neg a \vee \neg b))$	$\Rightarrow$	FALSE

- propositional formula = quantified formula with all variables existential

- using QBFs, several computational problems can be encoded more succintly than with propositional formulas, however

  - finding whether a formula is TRUE/FALSE is harder

  - model is not a set of literals but a *tree*



QBFs in DIMACS Format

- CNF DIMACS format extended with quantification information

1. `p cnf n1 n2`	n1 = max. variable index, n2 = num. of clauses
2. `e u1...un 0`	$\exists u_1, \ldots u_n$
3. `a v1 ...vm 0`	$\forall v_1, \ldots v_m$
...	
4. `l1 -l2 l3 0`	$l_1 \vee \neg l_2 \vee l_3$
...	

- $u_1, \ldots, u_n$ and $v_1, \ldots, v_m$ are natural numbers

- $l_1, -l_2, l_3$ are integers (negative number meaning a negated literal)

- whenever symbol `c` is seen, the rest of a line is treated as a comment

Delta Debugging

Skeletal delta debugging algorithm:

1. eliminate $\mathcal{C}_x$ clauses,
   - decrease number of clauses (n2) accordingly
   - if a variable no more used, remove it from the scoping information and normalize variables (n1) ??
2. if failure persists, update $\mathcal{C}_x$, and continue from 1.
3. try removing $\mathcal{C} \setminus \mathcal{C}_x$, if OK, update $\mathcal{C}_x$ and continue from 1,
4. otherwise, increase granularity (decrease $\mathcal{C}_x$).

Example

$$\forall a \exists b((a \vee b) \wedge (\neg a \vee \neg b))$$

- as a file (`a` and `b` encoded as 1 and 2, resp.):

`p cnf 2 2`	max variable idx = 2, 2 clauses
`a 1 0`	$\forall a$
`e 2 0`	$\exists b$
`1 2 0`	$a \vee b$
`-1 -2 0`	$\neg a \vee \neg b$

- given this file as an argument, a QBF solver (Quantor) prints:

```
s TRUE
c qnt (no variables exported)
```

Example continued

- removing first clause gives:

```
p cnf 2 1
a 1
e 2
c 1 2 0
-1 -2 0
```

- still TRUE (after all, you removed a constraint)
- continue by removing the second clause

```
p cnf 0 0
c a 1
c e 2
c 1 2 0
c -1 -2 0
```

- removing commented lines yields `p cnf 0 0` (which is TRUE)

Tools to Test Against

- `EBDDRES` available from `http://fmv.jku.at/ebddres`.
  - together with `qbv` (C. Wintersteiger, ETH Zürich)
  - `EBDDRES` may create an incorrect trace that `qbv` detects
- `Quantor` available from `http://fmv.jku.at/quantor`.
  - as with CNFs, you may eg. try to find the smallest FALSE instance

Grading

- project assignments should be chosen by 29.3.2007 23:59 CET

- For a passing grade of the programming assignment, you should:
  - choose target and implementation language and inform us,
  - prepare a written report (README) with installation instructions and source code and send it to us,
  - **DEADLINE:** 12.6.2007 16:00 CET,
  - book a demonstration time before end of semester.
  - All material should be sent to {`biere,toni.jussila`}`@jku.at`.