Model Checking of Fault-tolerant Systems

Lucas Baudin

June 15, 2016

Model Checking Modulo Theories

Model Checking

- a state type is a list of variables: $\mathbf{x}$
- a state is a valuation for these variables
- a transition is a formula over the current state variables and the next state variables
  (usually represented as a guard $H(\mathbf{x})$ and a (partial) assignment $V(\mathbf{x}, \mathbf{x}'))$
- $(H_1(\mathbf{x}) \wedge V_1(\mathbf{x}, \mathbf{x}')) \vee \ldots \vee (H_n(\mathbf{x}) \wedge V_n(\mathbf{x}, \mathbf{x}'))$

Modulo Theories

- the logical formula can be in any theory
- example: a system which has a variable x which keeps increasing unless it is lower than 0.
  - if the state type is a variable x of type int
  - transition: $(x < 0 \land x' = x) \lor (x \geq 0 \land x' = x + 1)$

```
if x < 0
  x' = x
if x >= 0
  x' = x + 1
```

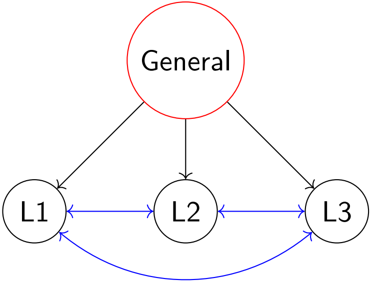
Model Checking of a Fault-tolerant System

The Byzantine General Problem

- One general wants to give an order to $n-1$ lieutenants. (Let's say that the general is just a special lieutenant.)
- Some of them may be faulty (including the general).
- In the end, they have to decide on the same order (at least for all the non faulty lieutenants).

An Algorithm

- Initial algorithm in (Lamport, Shostak, and Pease (1982))
- A simplified version



Pseudo-Code

```
% message[i][j] is the message sent by lieutenant j to i
message: array of (lieutenants * lieutenants) to message

source: the general id

% value[i] is the order that lieutenant i will follow
value: array of lieutenants to message

% good[i] is true if lieutenant i is non faulty
good: array of lieutenants to boolean
```

Pseudo-Code

```
propagate():
  forall i, j do
    if good[i]
      message[j][i] <- message[i][source]

agree():
  forall i do
    if good[i]
      value[i] <- majority(message[i])

propagate();
propagate();
agree();
```

The Byzantine General Problem

- $\exists v\ \forall i\ good[i] \Rightarrow value[i] = v$
- How many lieutenants can be faulty? Does that work for a third of the lieutenants?
- $\#\{i \mid good[i]\} > 2N/3$

The Byzantine General Problem

- $\exists v \; \forall i \; good[i] \Rightarrow value[i] = v$
- How many lieutenants can be faulty? Does that work for a third of the lieutenants?
- $\#\{i \mid good[i]\} > 2N/3$
- $\#\{i \mid good[i]\} > 2N/3 \Rightarrow \exists v \; \forall i \; good[i] \Rightarrow value[i] = v$

Sally

A model checker for infinite-state systems

- sri-csl.github.io/sally
- a symbolic model checker
- several engines: bmc, kind, ic3
- works with various smt solvers: mathsat, yices2, z3

Input Language

- lisp-like language
- low level
- easy to parse and work with

Input Language

- state type

```
(define-state-type my_state_type
  ((x Real) (y Real))
)
```

Input Language

- state type

```
(define-state-type my_state_type
  ((x Real) (y Real))
)
```

- state formula

```
(define-states x_is_zero my_state_type
  (= x 0)
)
```

Input Language

- transition: a first order formula over state variables and next state variables

```
(define-transition my_transition my_state_type
  (or
    (= next.x (+ state.x 1))
    next.x_is_zero
  )
)
```

Input Language

- queries: check that a property always holds

```
(query my_system
  (>= x 0)
)
```

A new old input language: Sal

Sal

- an older model checker, developed at SRI
- developed actively until 2006, minor versions until 2013
- finite state systems

Input language

- already used
- supports modules, composition

Input language

```
my_module: MODULE
BEGIN
  OUTPUT
    x: REAL,
    y: REAL
  INITIALIZATION
    x = 0;
  TRANSITION
    ...
END;
```

```
TRANSITION
  [x >= 0 -->
     x' = x + 1;
     y' IN { i: REAL | TRUE }
  []
   TRUE -->
     x' = 0;
     y' IN { i: REAL | TRUE }
  ]
```

Input language

- a lemma is translated to a Sally query
- multiple lemma
- syntax for temporal logic (not available in Sally)

```
my_context: CONTEXT =
BEGIN
  my_module: MODULE
       ...

  always_positive: LEMMA
    my_module |- G(x >= 0);

  wrong_lemma: LEMMA
    my_module |- G(x > 0 -> x = 1)

END
```

Parametrization

Parametrization

- The Byzantine Generals problem can be solved using existing tools. . .

Parametrization

- The Byzantine Generals problem can be solved using existing tools. . .

. . . but only for a fixed number of lieutenants.

- adding n more variables for n new processes does not scale and does not provide a general proof.

Arrays

- arrays indexed by process
- used in Cubicle (Conchon et al. (2012)), a parametrized model checker which supports a fragment of arrays theory
- depending on the fragment of the theory used, might be very hard to verify

Quantifiers

- for most examples, they can be avoided in transitions
- works only with z3
- example:

```
(forall (i Int) (select a i))
```

Counting in SMT

- $\phi(y) \wedge y = \# \{x | \psi(x)\}$
- $\psi(.)$: first order formula of Presburger arithmetic, then with arrays too
- how is it solved?

State of the art

- Bradley, Manna, and Sipma (2006): a decision procedure for a fragment of arrays, with distinct theories for elements and indexes
- Alberti, Ghilardi, and Pagani (2016): a decision procedure for counting on arithmetic and arrays, via various rewriting and quantifier eliminations, mix index and elements theories, but quantify only over one element at a time
- Bjørner, Gleissenthall, and Rybalchenko (n.d.): a model checking oriented way to deal with arrays (one update at a time for every arrays)

Counting over Presburger arithmetic

- given a model, one can compute the value of $\#\{x|\psi(x)\}$
- given an ordering on the integer variables, one can compute the symbolic value of $\#\{x|\psi(x)\}$
- $\Rightarrow$ symbolic computation of cardinalities, for the ordering of a given model

Counting over Presburger arithmetic, with an ordering

- the ordering is called an oracle: when asked wether $a > b$, it looks in the model the value of a and b and answers accordingly.
- when the cardinality is computed symbolically, it is equal to a formula which holds under some assumptions, and the oracle can say what they are

Counting over Presburger arithmetic, with an ordering

- the ordering is called an oracle: when asked wether $a > b$, it looks in the model the value of a and b and answers accordingly.
- when the cardinality is computed symbolically, it is equal to a formula which holds under some assumptions, and the oracle can say what they are

- example: $\phi(y) \wedge y = \#\{x | 0 \leq x < z \wedge 0 \leq x < u\}$
- if the oracle says $z > u$, then y can be computed and $y = z$.
- under the assumption $z > u$, $y = z$
  - $z > u \Rightarrow y = z$
  - $z \leq u \Rightarrow y = u$

Counting over Presburger arithmetic, with an ordering

- compute a symbolic interval list in which the formula is satisfiable i.e. $\exists I \in V_x(\psi)\ x \in I \Leftrightarrow \psi(x)$
- if members of $V_x(\psi)$ are disjoint $card(V_x(\psi)) = \sum_{[v,v'] \in V_x(\psi)} (v - v') = \#\{x | \psi(x)\}$

Counting over Presburger arithmetic, with an ordering

- compute a symbolic interval list in which the formula is satisfiable i.e. $\exists I \in V_x(\psi)\ x \in I \Leftrightarrow \psi(x)$
- if members of $V_x(\psi)$ are disjoint $card(V_x(\psi)) = \sum_{[v,v'] \in V_x(\psi)} (v - v') = \#\{x | \psi(x)\}$
- $V_x(y < x) = \{[y, +\infty)\}$

Counting over Presburger arithmetic, with an ordering

- $V_x(\psi \wedge \phi) = V_x(\psi) \sqcap V_x(\phi)$
  where $A \sqcap B$ is the set of every intersection of an interval of A with an interval of B (of course some are empty and must be deleted)

Counting over Presburger arithmetic, with an ordering

- $V_x(\psi \wedge \phi) = V_x(\psi) \sqcap V_x(\phi)$
  where $A \sqcap B$ is the set of every intersection of an interval of A with an interval of B (of course some are empty and must be deleted)

- Intersection between two intervals: $[a, b)$ and $[c, d)$ can be computed: there is an oracle giving the ordering on the variables, so $max(a, c)$ and $min(b, d)$ are computable given the assumptions that the oracle makes.
  Then the intersection is $[max(a, c), min(b, d))$ if $max(a, c) \leq min(b, d)$

Counting over Presburger arithmetic, with an ordering

- if $V_x(\psi) = \{[a_1, b_1), \ldots [a_n, b_n)\}$ (with $a_1 \leq b_1 \leq \ldots \leq b_n$, $a_1 \neq -\infty$ and $b_n = +\infty$)
- $V_x(\neg\psi) = \{(-\infty, a_1), [b_1, a_2), \ldots, [b_n, +\infty)\}$
- other cases are easy too, disjunction on $a_1 = -\infty$ and $b_n = +\infty$

Counting with multiplication

- to deal with constant multiplications, a modulo information can be added to every intervals (such as $([5, 10), = 1[3])$ are the integers x between 5 and 10 and such that $3|x - 1$)
- intersection, negation of these intervals can be done in an analog way

Counting in SMT

- this problem is harder than universal quantification or existential quantification
- this algorithm is exponential (if for every set of assumptions, it turns out that the cardinality constraints is not satisfiable, everyone of them must be checked)
- if there is few constraints on the variables in the counting constraints, and that the problem is unsat, it is unpractical

Future Work

- Express lower/upper bounds for counting constraints instead of the precise symbolic expression (might speed up when the problem is unsat).
- Counting over arrays
- IC3 with arrays and counting quantifiers.

References

Alberti, Francesco, Silvio Ghilardi, and Elena Pagani. 2016. “Counting Constraints in Flat Array Fragments.” *CoRR* abs/1602.00458. `http://arxiv.org/abs/1602.00458`.

Bjørner, Nikolaj, Klaus v Gleissenthall, and Andrey Rybalchenko. n.d. “Cardinalities and Universal Quantifiers for Verifying Parameterized Systems.”

Bradley, Aaron R, Zohar Manna, and Henny B Sipma. 2006. “What’s Decidable About Arrays?” In *Verification, Model Checking, and Abstract Interpretation*, 427–42. Springer.

Conchon, Sylvain, Amit Goel, Sava Krstic, Alain Mebsout, and Fatiha Zaïdi. 2012. “Cubicle: A Parallel SMT-Based Model Checker for Parameterized Systems - Tool Paper.” In *CAV*, edited by P. Madhusudan and Sanjit A. Seshia, 7358:718–24. Lecture Notes in Computer Science. Springer.

Lamport, Leslie, Robert Shostak, and Marshall Pease. 1982. “The