nuXmv: infinite state model checking

Enrico Magnago

University of Trento,
Fondazione Bruno Kessler

Infinite state transition system

Background

Are you all familiar with computability concepts?

- decidability,
- undecidability,
- reduction.

Background

Are you all familiar with computability concepts?

- decidability,
- undecidability,
- reduction.

What about computational models?

- Turing machine,
- 3-counter machine,
- 2-counter machine.

Finite is nice

Finite models

- Up until now we have seen only finite models: representable as finite graphs.
- Nice theoretical results: *decidability* for both reachability and liveness.
- Sound and complete procedures: if a counter-example exists, then there exist also a looping counter-example.

Why infinite?

Sources of infinity

- **data manipulation**: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$;
- **control structures**: procedures, process creation;
- **async communication**: unbounded FIFOs;
- **parameterised models**: check correctness for all possible parameters;
- **time**: timed/hybrid systems;
- ....

Infinite transition systems

Infinite state

- Represent an infinite graph (infinite number of states).
- There might be no looping counter-example, does **not** imply that the property holds. **Why?**

Where are the problems?

- $M \models \psi$
- $\mathcal{L}(M) \subseteq \mathcal{L}(T_\psi)$
- $\mathcal{L}(M) \cap \mathcal{L}(T_{\neg\psi}) = \emptyset$
- $\mathcal{L}(M \times T_{\neg\psi}) = \emptyset$

A nice pair of innocent looking types

nuXmv

supports the description of infinite state transition systems through the types: `integer` ($\mathbb{Z}$) and `real` ($\mathbb{R}$).

Here be dragons

Reachability

- Can you we define a reduction from the halting problem of a 2-counter machine to reachability on an infinite state transition system?

Here be dragons

Reachability

- Can you we define a reduction from the halting problem of a 2-counter machine to reachability on an infinite state transition system?
- What can we conclude?

Liveness

- Can you we define a reduction from the halting problem of a 2-counter machine to $\mathrm{LTL}/\ \mathrm{CTL}$ checking on an infinite state transition system?

Here be dragons

Reachability

- Can you we define a reduction from the halting problem of a 2-counter machine to reachability on an infinite state transition system?
- What can we conclude?

Liveness

- Can you we define a reduction from the halting problem of a 2-counter machine to $\mathrm{LTL}/\ \mathrm{CTL}$ checking on an infinite state transition system?
- What can we conclude?

Reachability

- Can you we define a reduction from the halting problem of a 2-counter machine to reachability on an infinite state transition system?
- What can we conclude?

Liveness

- Can you we define a reduction from the halting problem of a 2-counter machine to $\mathrm{LTL}/\ \mathrm{CTL}$ checking on an infinite state transition system?
- What can we conclude?

Invariant, LTL and CTL checking are undecidable.

But Timed automata...

Timed Automata

- Infinite state transition system ($\texttt{clock} \in \mathbb{R}$).
- Reachability on timed automata is decidable, why?

But Timed automata...

Timed Automata

- Infinite state transition system ($\texttt{clock} \in \mathbb{R}$).
- Reachability on timed automata is decidable, why?
- Bisimulation with a finite state transition system: *region-abstraction*.

Verification on infinite state systems

BMC and K-induction

- BMC and K-induction can be *trivially* extended to infinite-state systems.
- They are still sound, we loose completeness:
    - **BMC** : look for looping counter-example,
    - **K-induction** : bad state reachable by infinite run without initial states.
- Use an SMT-solver able to handle required theories: *integers*, *reals*.

Other techniques have been adapted for infinite-state systems: *liveness-to-safety*, *ic3*, *abstract-interpretation* . . . .

Exercises

Equality using 2-counter machine [1/6]

Check if the 2 counters contain the same value

- Write a program for a 2-counter machine that decides whether the counters contain the same value.
- Model this program in nuXmv.
- Prove termination and correctness.

Equality using 2-counter machine [2/6]

pseudocode

```
while(true):
  if c0 == 0:
    return c1 == 0
  if c1 == 0:
    return c0 == 0
  c0--;
  c1--;
```

Equality using 2-counter machine: translate in SMV [3/6]

Straightforward translation into SMV

```
MODULE main
VAR
  c0 : integer;
  c1 : integer;
  l : {check, decr_c0, decr_c1,
       end_equal, end_not_equal};

INVAR c0 >= 0 & c1 >= 0;
```

Equality using 2-counter machine: translate in SMV [4/6]

```
ASSIGN
  init(l) := check;
  next(l) :=
    case
      l = check & c0 = 0 & c1 = 0  : end_equal;
      l = check & c0 = 0 & c1 != 0 : end_not_equal;
      l = check & c0 != 0 & c1 = 0 : end_not_equal;
      l = check : decr_c0;
      l = decr_c0 : decr_c1;
      l = decr_c1 : check;
      l = end_equal : end_equal;
      l = end_not_equal : end_not_equal;
    esac;
```

Equality using 2-counter machine: translate in SMV [5/6]

```
ASSIGN next(c0) :=
  case
    l = decr_c0 & c0 > 0 : c0 - 1;
    TRUE : c0;
  esac;

ASSIGN next(c1) :=
  case
    l = decr_c1 & c1 > 0 : c1 - 1;
    TRUE : c1;
  esac;
```

Equality using 2-counter machine: properties [6/6]

Properties

- The *end* states are reachable.

Equality using 2-counter machine: properties [6/6]

Properties

- The *end* states are reachable.

  ```
  INVARSPEC l != end_equal;
  INVARSPEC l != end_not_equal;
  ```
- Every execution terminates.

Equality using 2-counter machine: properties [6/6]

Properties

- The *end* states are reachable.

  ```
  INVARSPEC l != end_equal;
  INVARSPEC l != end_not_equal;
  ```

- Every execution terminates.

  ```
  LTLSPEC F (l in {end_equal , end_not_equal});
  ```

  What happens when we ask the system to verify this property?
  Do you have an intuition about why this is the case?

Thermostat

Model

- a thermostat has 2 states: *on* and *off*.
- at every clock tick the thermostat checks the current temperature:
  - if the temperature is below 18 degrees the thermostat switches *on*.
  - if the temperature is above 18 degrees the thermostat switches *off*.
- when the thermostat is *off* the temperature drops; the drop in temperature is at most *max_dt* degrees.
- when the thermostat is *on* the temperature increases; the increase in temperature is at most *max_dt* degrees.
- the temperature initially is in $[18 - max\_dt; 18 + max\_dt]$.

Prove that the thermostat keeps the temperature in the range $[18 - max\_dt; 18 + max\_dt]$, for all *max_dt* in $\mathbb{R}$.

Thermostat: solution

```
MODULE main
DEFINE threshold := 18;
FROZENVAR max_dt : real;
VAR
  temperature : real;
  state : {on, off};

INIT temperature >= threshold - max_dt;
INIT temperature <= threshold + max_dt;

INVAR temperature < threshold -> state = on;
INVAR temperature > threshold -> state = off;

TRANS state = off -> next(temperature) < temperature &
                     next(temperature) >= temperature - max_dt;
TRANS state = on  -> next(temperature) > temperature &
                     next(temperature) <= temperature + max_dt;

INVARSPEC temperature >= threshold - max_dt &
          temperature <= threshold + max_dt;
```