Model Checking with MCMAS

Natasha Alechina   Brian Logan

Utrecht University

n.a.alechina@uu.nl   b.s.logan@uu.nl

Outline of this lecture

- practical model checking
- overview of ISPL
- model checking examples

Practical model checking

A very brief introduction to model checking

Formal verification consists of three parts:

- a description language for describing the system to be verified
- a specification language for describing the properties to be verified
- a verification method to establish whether the description of the system satisfies the specification

Proof-based approaches to verification

- the system description is a set of formulas Γ in some logic
- the specification is another formula φ in the same logic
- the verification method consists of trying to find a proof that $\Gamma \vdash \varphi$
- this is time consuming and requires expertise on the part of the user

Model-based approaches to verification

- the system is represented by a finite model M for an appropriate temporal logic
- the specification is a formula φ in the same logic
- the verification method consists of computing whether $M \models \varphi$
- this process can be automated using a model checking algorithm

Verifying properties by model checking

To verify that a program or system satisfies a property, we

- describe the system using the description language of the model-checker
- express the property to be verified using the specification language of the model checker (with many model checkers, the specification is given directly in temporal logic)
- run the model checker with the system description and property to be verified as inputs

How it works

When the model checker is run

- it generates a model (transition system), M, from the system description
- if necessary, converts the property to be verified into a temporal logic formula φ
- for every state $s \in M$, checks whether s satisfies φ, i.e., $M, s \models \varphi$
- if the model doesn't satisfy the formula, most model checkers output a counterexample trace of the system behaviour that causes the failure

Symbolic model checking

- the system description is usually fairly compact, but the search space may be *very* large (we'll return to this is later lectures)
- explicit state model checkers exhaustively enumerate all states in the search space — this is only practical for small systems
- symbolic model checkers use clever encodings of the serarch space (e.g., BDDs) in which *sets of states* are represented by a boolean formula
- allows compact representation of large search spaces, e.g,. up to 10^{20} states

Aside: program model checking

- model checkers take as input a (usually abstract) description of the system or program to be verified
- how do we know the system description is correct?
- in some cases, it is possible to automatically transform, e.g., an agent program, into a system description in a provably correct way; see, e.g., Doan et al. Verifying Heterogeneous Multi-agent Programs, AAMAS 2014, 149–156
- another approach is to use program model checking, which uses the program text, e.g., Java bytecodes, as the system description; see, e.g., Java PathFinder developed by NASA
- program model checking ensures there is no mismatch between the system and its description, but only small systems can be checked

MCMAS

- different model checkers support different temporal logics and different description languages
- we will use the model checker MCMAS, originally developed by Franco Raimondi and Alessio Lomuscio at Imperial College
- it supports CTL, CTLK, ATL and ATLK[1]
- the system description language is ISPL
- ISPL allows specification of systems consisting of multiple agents, each with their own beliefs, actions they can perform etc.

[1] Version 1.3.0 also supports LTL, LDL, and CTL*.

Overview of ISPL

ISPL files: agents

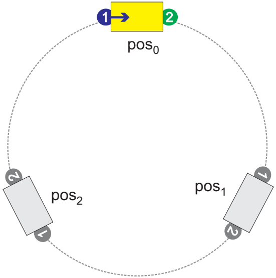
- one or more `Agent` sections containing the specification of the agents in the system (and the environment, if there is one)
  - a `Vars` section that implicitly defines the possible *local states* of the agent
  - an `Actions` section specifying the actions the agent can perform
  - a `Protocol` section specifying which actions can be performed in which local states of the agent
  - an `Evolution` section specifying how the local state of the agent is updated as a result of the actions performed by all agents

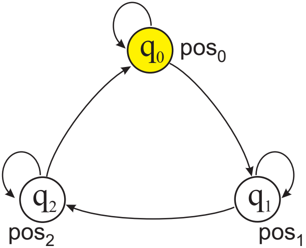
ISPL files: everything else

- an `Evaluation` section defining propositions specifying properties of the MAS in terms of the local states of the agents
- an `InitStates` section specifying the initial states of the MAS
- a `Formulae` section defining the properties to be verified
- see the ISPL tutorial on Blackboard for details

Model checking examples

Example: robot and carriage





Example: robot and carriage 1

```
-- INFOMLSAI 2021 Example ISPL file ...
Agent Robot1
    Vars:
        pos: {pos0, pos1, pos2};
    end Vars
    Actions = {push1, nil};
    Protocol:
        pos=pos0 or pos=pos1 or pos=pos2: {push1, nil};
    end Protocol
    Evolution:
        pos=pos0 if pos=pos0 and Action = nil;
        pos=pos0 if pos=pos2 and Action = push1;
        pos=pos1 if pos=pos1 and Action = nil;
        pos=pos1 if pos=pos0 and Action = push1;
        pos=pos2 if pos=pos2 and Action = nil;
        pos=pos2 if pos=pos1 and Action = push1;
    end Evolution
end Agent
```

Example: robot and carriage 2

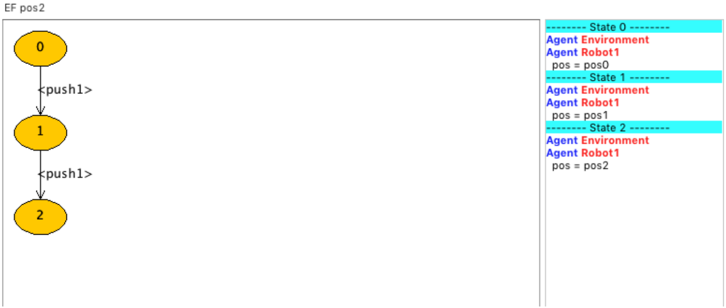
```
Evaluation
    pos0 if Robot1.pos  = pos0;
    pos1 if Robot1.pos  = pos1;
    pos2 if Robot1.pos  = pos2;
end Evaluation

InitStates
    Robot1.pos = pos0;
end InitStates

Formulae
    EF pos2;
    A (pos0 U pos1);
    AG (pos0 or pos1 or pos2);
end Formulae
```

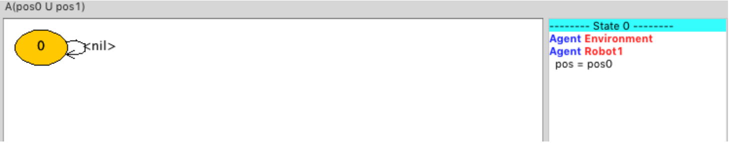
Property *EF* pos2

- there exists a path where in the future pos2 is true, i.e., the carriage is in pos_2
- this formula is true in the model, and we can ask MCMAS for a witness — a path on which the formula holds



Property *A* pos0 *U* pos1

- on all paths pos0 is true until pos1 becomes true
- this formula is false in the model, and we can ask MCMAS for a counterexample — a path on which the formula does not hold



Property $AG\,(\text{pos0} \vee \text{pos1} \vee \text{pos2})$

- on all points on all paths pos0 $\vee$ pos1 $\vee$ pos2 is true
- this formula is true in the model
- however there is no witness smaller than the model itself (as the formula has to hold at all points on all paths)
- model checkers can only generate witnesses for existential formulae and counterexamples for universal formulae