Model Checking

In Industry,Efficiency, CTL, Fairness Constraints, LTL

Ranald Clouston

May 22, 2025

Australian National University

Model Checking

Given a model, and a logical formula / proposition, does that model satisfy that proposition?

- A yes/no question; perhaps in the case of 'no' we would like an explanation

In principle this question could be asked of any logic with semantics:

- Propositional logic: Given an assignment of truth values to propositional variables, is a proposition true?
- First order logic: Given a universe of discourse, interpretation of functions and relations, and an environment assigning universe elements to variables, is a formula satisfied?

But the most significant applications of model checking have involved temporal logic:

- Give a transition system, and a path / state, is this LTL / CTL / CTL* proposition satisfied?

Model Checking in Industry

The Pentium Bug

The notorious Pentium bug (sometimes called the FDIV bug) of 1994 was a subtle hardware error in an Intel microprocessor, effecting floating point computations.

- Discovered by computational number theorist Thomas R. Nicely

Intel initially argued the bug was not significant

- Estimate that 1 in 9,000,000,000 computations was effected

But correct hardware is literally a life and death matter in some cases.

- Eventual recalls cost Intel half a billion US dollars (in 1994 money!), not to mention reputational cost

Formal Methods

Formal methods is mathematical proof of properties of computer systems.

- As opposed to testing (though testing will always be used alongside)
- A high level of rigour comes with a cost in time, logicians, and money...
- but getting it wrong can be costly also!

Avoiding billion dollar recalls is clearly a potential motivator for formal methods

- In general, how bad is it if this system fails to act as it should?

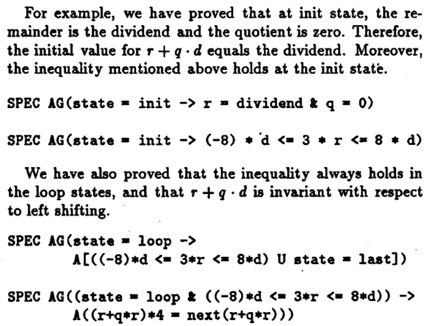
Many real computer systems are much too big to tackle with the pen-and-paper logical techniques we are using in this course.

- Automation is necessary!
- Understanding the pen-and-paper techniques is the foundation of understanding what the automatic logical tools are trying to accomplish.

Formal Methods at IBM

The mid 1990s saw IBM emerge as a leader in formal methods for hardware

- Key technique: model checking with CTL
- e.g. the 1996 papers Word Level Model Checking - Avoiding the Pentium FDIV Error and RuleBase: an Industry-Oriented Formal Verification Tool:



For example, we have proved that at init state, the remainder is the dividend and the quotient is zero. Therefore, the initial value for $r + q \cdot d$ equals the dividend. Moreover, the inequality mentioned above holds at the init state.

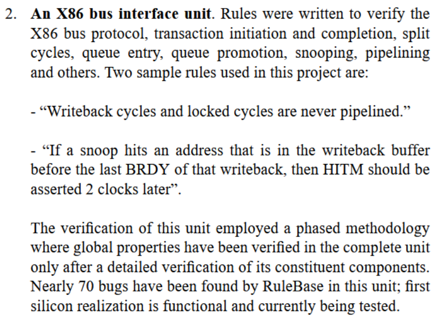
```
SPEC AG(state = init -> r = dividend & q = 0)

SPEC AG(state = init -> (-8) * d <= 3 * r <= 8 * d)
```

We have also proved that the inequality always holds in the loop states, and that $r + q \cdot d$ is invariant with respect to left shifting.

```
SPEC AG(state = loop ->
        A[((-8)*d <= 3*r <= 8*d) U state = last])

SPEC AG((state = loop & ((-8)*d <= 3*r <= 8*d)) ->
        A((r+q*r)*4 = next(r+q*r)))
```



2. **An X86 bus interface unit**. Rules were written to verify the X86 bus protocol, transaction initiation and completion, split cycles, queue entry, queue promotion, snooping, pipelining and others. Two sample rules used in this project are:

   - "Writeback cycles and locked cycles are never pipelined."

   - "If a snoop hits an address that is in the writeback buffer before the last BRDY of that writeback, then HITM should be asserted 2 clocks later".

   The verification of this unit employed a phased methodology where global properties have been verified in the complete unit only after a detailed verification of its constituent components. Nearly 70 bugs have been found by RuleBase in this unit; first silicon realization is functional and currently being tested.

Industrial Model Checking Now

Model checking has continued to be applied and developed. Examples from the last decade:

- Applying Model Checking to Industrial-Sized PLC Programs: programs for Programmable Logic Controllers at CERN (European Organization for Nuclear Research): "the size of the potential state space (PSS) is 1.6×10^{218}"
- Automated test generation using model checking: an industrial evaluation: software analysis for German manufacturer Bombardier Transportation.
- Practical applications of model checking in the Finnish nuclear industry: "Since 2008, VTT has applied model checking in practical customer work related to the Olkiluoto 3 EPR, Loviisa 1&2 VVER-440, and Hanhikivi 1 AES-2006 nuclear power plants... The verified properties are specified using temporal logic languages such as Linear Temporal Logic (LTL) or Computational Tree Logic (CTL)"

Computational Efficiency of Model Checking Temporal Logic

Considering Efficiency

In this course we will not formally analyse the time and space costs of algorithms.

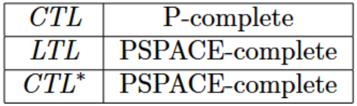
- But cost considerations are critical to the real world use of implemented algorithms.

We have a variety of tools for reasoning about the cost of algorithms, and the innate cost of problems (how good could be the theoretically best algorithm that solves this problem be?)

- Testing by timing.
- Expressing the time or space used as a function of the size of the input(s), in worst / best / average etc. cases.
- Big O notation expresses these functions in a cruder but more robust manner.
- Complexity classes group problems together that have similar costs.

The Complexity of Model Checking

An analysis of the problems (not merely an analysis of particular algorithms!):



CTL	P-complete
LTL	PSPACE-complete
*CTL**	PSPACE-complete

Figure: Ph. Schnoebelen, The Complexity of Temporal Logic Model Checking (2002)

- CTL model checking solvable in time proportional to a polynomial of its inputs
- LTL and CTL* model checking in space proportional to a polynomial of its inputs

These may not look like big differences, but no known algorithm exists for a PSPACE-complete problem that does not take exponential time.

- Technically, $P \subseteq NP \subseteq PSPACE$ and it is generally believed that, at least, $P \neq NP$, but no one has yet been able to prove this!

Measuring Complexity of Model Checking Problems

Model checking problems have two inputs: model and proposition.

- Measure a model by separately measuring the number of states, and of transitions;
- Measure a proposition by its number of connectives;
- A problem or algorithm might be e.g. exponential in one of these measurements and only linear in the other; the worse cost will tend to dominate the better and therefore be used as the 'headline' figure.

A related but separate question: can we restrain the cost of our problem / algorithm by keeping our model and/or proposition small?

- Outside the scope of this course, but motivates work on e.g. Binary Decision Diagrams.

Model Checking CTL

Model Checking via Labelling States

We will see an algorithm for model checking vs the connectives $\bot, \neg, \wedge, EX, EU, EG$

- We have seen that this set is adequate for CTL, so simplifies our algorithm
- Even this gives rise to a practical question: what is the cost of converting a CTL formula to this 'smaller' logic?

Basic idea: label every state (not just the start state!) in our system by every subformula of our proposition that it satisfies

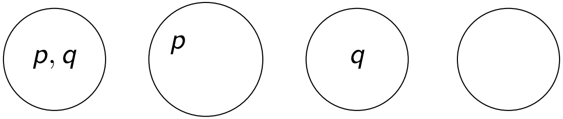
- $\mathit{subformulas}(\bot) = \bot$ and $\mathit{subformulas}(p) = \{p\}$
- $\mathit{subformulas}(\neg\varphi) = \{\neg\varphi\} \cup \mathit{subformulas}(\varphi)$, and EX, EG likewise.
- $\mathit{subformulas}(\varphi \wedge \psi) = \{\varphi \wedge \psi\} \cup \mathit{subformulas}(\varphi) \cup \mathit{subformulas}(\psi)$, and EU likewise

We will work bottom up, labelling with all proper subformulas of φ before we label anything with φ.

Labelling with Propositional Logic

- Label with propositional variables according to the labelling function;
- Label nothing with $\perp$;
- If a state is not labelled with φ, label it with $\neg\varphi$;
- If a state is labelled with both φ and ψ, label it with $\varphi \wedge \psi$.

e.g. formula $p \wedge \neg q$:

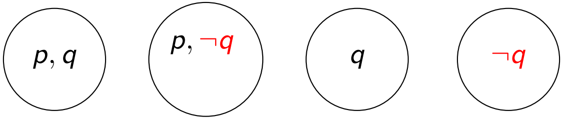


(transition arrows left off because they are not relevant to these labels)

Labelling with Propositional Logic

- Label with propositional variables according to the labelling function;
- Label nothing with $\perp$;
- If a state is not labelled with φ, label it with $\neg\varphi$;
- If a state is labelled with both φ and ψ, label it with $\varphi \wedge \psi$.

e.g. formula $p \wedge \neg q$:

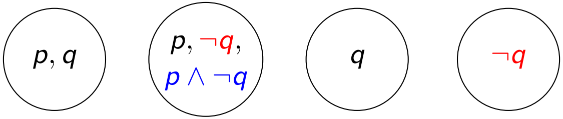


(transition arrows left off because they are not relevant to these labels)

Labelling with Propositional Logic

- Label with propositional variables according to the labelling function;
- Label nothing with $\bot$;
- If a state is not labelled with φ, label it with $\neg\varphi$;
- If a state is labelled with both φ and ψ, label it with $\varphi \wedge \psi$.

e.g. formula $p \wedge \neg q$:



(transition arrows left off because they are not relevant to these labels)

Labelling with Exists neXt

- Label a state with $EX\varphi$ if any of its successors are labelled with φ.

Whiteboard example: take a one-state system with an empty labelling function and label it for the proposition $\neg EXp \to EX\neg p$.

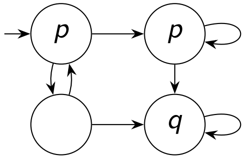
Labelling with Exists Until

Give a proposition $E[\varphi\, U\, \psi]$, and labelling completed for φ and ψ,

- Label with $E[\varphi\, U\, \psi]$ all states that are labelled with ψ;
- Label with $E[\varphi\, U\, \psi]$ all states that are labelled with φ for which some immediate successor is labelled with $E[\varphi\, U\, \psi]$;
- Repeat the second step until there are no more states than can be so labelled.

To avoid ever visiting the same state twice, search backwards from $E[\varphi\, U\, \psi]$ states.

Whiteboard example: $E[p\, U\, q]$ and

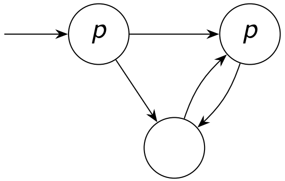


Labelling with Exists Globally: A First Attempt

Given a proposition $EG\varphi$, and labelling completed for φ,

- Label with $EG\varphi$ all states that are labelled with φ;
- *Delete* $EG\varphi$ from any state for which *none* of its successors are labelled with $EG\varphi$;
- Repeat the second step until no more such deletions are possible.

Whiteboard example: EGp and



Improving the Exists Globally Algorithm

The *EG* algorithm of the previous slide is simple and good enough for on-paper work, but it sometimes considers the same state more than once, which could be inefficient.

A more sophisticated version of the algorithm involves a concept called a strongly connected component (SCC):

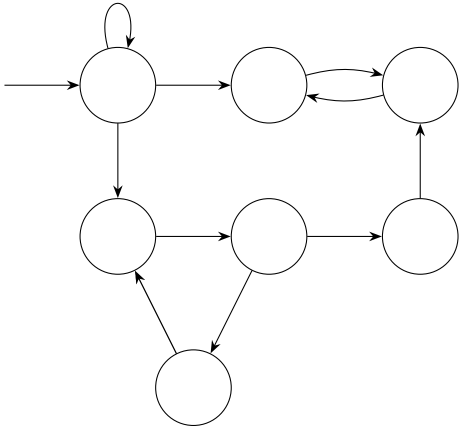
- A set of states that can all reach each other in zero or more steps...
- that is *maximal*: cannot be expanded to a bigger set that keeps this property

An SCC is trivial if it contains one state and no self-loop.

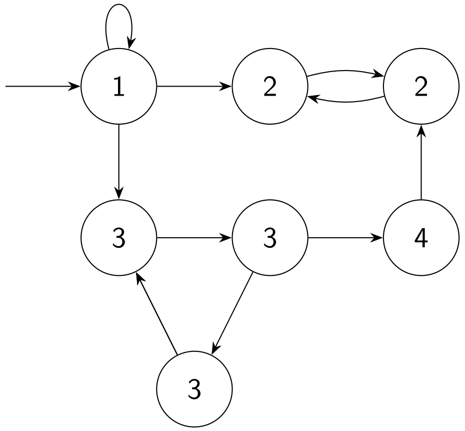
- Intuition: a system can get stuck in a non-trivial SCC forever.

We will not give an explicit algorithm, but SCCs can be identified in linear time.

SCC Example



SCC Example



Only the SCC numbered 4 is trivial.

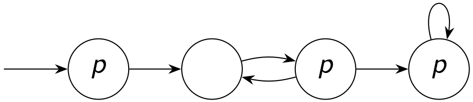
Exists Globally, with SCCs

Given a proposition $EG\varphi$, and labelling completed for φ,

- Ignore (temporarily) all states not labelled φ;
- Identify the SCCs in this subgraph;
- Label with $EG\varphi$ any state in the subgraph that can reach a non-trivial SCC in any number of transitions (including any state in a non-trivial SCC).

Whiteboard examples

- The example from our first attempt at an EG algorithm;
- EGp with



Analysis of our Model Checking Algorithm

Our algorithm considers each connective in our proposition once, so is linear in the number c of connectives.

Per connective, it considers each state (and each transition) at most once, so for each connective it is linear in the numbers s of states and t of transitions.

So time complexity is $O(c \times (s + t))$

- Triple the size of the proposition, cost approximately triples;
- Triple the size of the transition system, cost approximately triples;
- But there is a multiplication, so triple both and cost approximately up by factor of nine.

CTL with Fairness Constraints

Pathological Paths

There are sometimes paths through a system that are so unrealistic that we do not care about them for the purpose of verification.

- e.g. pathological user behaviour, like an elevator system where users go up but no user ever requests to go down.

This creates 'false negatives': failed model checks which do not actually indicate a 'real' bug.

How to avoid these false negatives?

- Fix our system to rule out these paths? Good idea! But maybe difficult or impossible.
- Fix our proposition with an implication to say that we are only interested in certain paths. Good idea! But what if the condition is not expressible in our logic of choice?

Fairness Constraints

The model checker NuSMV has special support for just such a situation:

- CTL model checking restricted to paths obeying special propositions called fairness constraints which are not expressible in CTL.
- Fairness constraints are CTL propositions asserted to hold *infinitely often*.
- Of course 'infinitely often' *is* expressible in LTL, as GF

'Occurs infinitely often' is a common thing to specify because it describes any activity that is expected to keep happening across the entire lifetime of the system, without committing to how often it will happen.

- As long as the elevator is operational, there will be people pressing every available button in the future of its operation.

Fair Paths

Fix a set C of CTL propositions we will call fairness constraints.

A fair path is a path $s_0 \to s_1 \to s_2 \to \cdots$ in a model $\mathcal{M}$ for which for all fairness constraints φ, there are infinite many natural numbers i, such that $\vDash_{\mathcal{M},s_i} \varphi$.

We then introduce new connectives $E_C X, E_C G, E_C U$, with new semantics:

- $\vDash_{\mathcal{M},s} E_C G \varphi$ if there exists a **fair** path from s whose every state satisfies φ.

$E_C X \varphi$ will simply mean $EX(\varphi \wedge E_C G \top)$, and $E_C[\varphi \, U \, \psi]$ means $E[\varphi \, U \, \psi \wedge E_C G \top]$

- So no need for more new semantics or model checking rules once we handle $E_C G$.
- Recall $\top$ is any theorem, e.g. $\neg\bot$.

Model Checking the Fair version of Exists Globally

A fair strong connected component is a non-trivial SCC in which all fairness constraints appear at some state

- Intuition: a system can get stuck in a non-trivial SCC forever, and visit each fairness constraint infinitely often while there.

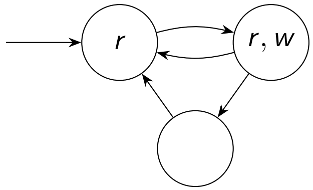
We then need only a mild edit to our algorithm for EG:

Given a proposition $E_C G\varphi$, and labelling completed for φ and for all propositions in C,

- Ignore (temporarily) all states not labelled φ;
- Identify the SCCs in this subgraph;
- Label with $E_C G\varphi$ any state in the subgraph that can reach a **fair** SCC in any number of transitions (including any state in a fair SCC).

Example

Show that $EG(w \vee EXw)$ holds for



But let our constraints C be $\{\neg r\}$. Show then that $E_C G(w \vee E_C Xw)$ does not hold.

Analysis of Model Checking with Fairness

Formally, we have defined a new logic, CTL with fairness constraints, which is more expressive than CTL.

What is the impact on efficiency?

We do need to some more work - to model check each of our fairness constraints - but we are linear in the size of these constraints, and remain linear in our other inputs.

- So despite dipping our toes into LTL territory with ‘occurs infinitely often’, we do not inherit the bad computational properties of LTL.

Model Checking LTL

From States to Paths in Model Checking

The state labelling approach does not seem to be extendable to LTL, because LTL is about paths, not states

- A state might have successors with p, and without; do we label it Xp?

Solution: Build a bigger transition system that makes copies of states of the original model, which are then labelled with different collections of non-variable propositions.

- e.g. two versions of a state, one with Xp, one without.
- If a state is labelled with φ in the big transition system, all paths from it satisfy φ.
- So if φ is the proposition we are checking, then the model check succeeds if and only if all copies of the start state are labelled with φ.

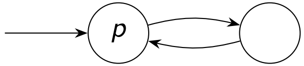
States of the New Transition System

Given a transition system $\mathcal{M}$ and proposition φ, we start by ignoring the transition relation and setting out some possible states of our bigger transition system.

All copies of a state s should agree with s on variables.

We make as many copies as we need to make labels with all different combinations of the subformulas of φ, except those that cause a contradiction.

Easy example: proposition XXp and transition system



Non-Contradictory Labels

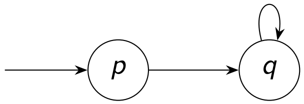
We will use only connectives $\bot, \neg, \wedge, X, U$ (all others are expressible).

Rules for which subformulas we can legally put together in states:

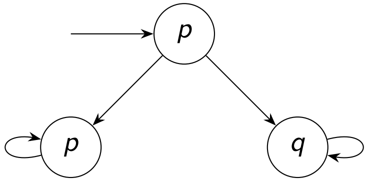
- No state gets label $\bot$.
- A state gets label $\neg\varphi$ if and only if it does not get label φ (recall that absence of a proposition means exactly that its negation holds).
- A state gets label $\varphi \wedge \psi$ if and only if it gets both φ and ψ.
- Neither $X\varphi$ nor its absence causes a contradiction, so we need state copies with each.
- Add $\varphi\, U\, \psi$, making a copy if necessary, unless a state has neither φ nor ψ.
- Leave out $\varphi\, U\, \psi$, making a copy if necessary, unless a state has ψ.

More Examples

$Xp \; U \; q$ and



$p \; U \; q$ and



Transitions

Given states s', t', which are copies of s, t from the original model, add a transition $s' \to t'$ if

- There is a transition $s \to t$ in the original model, and
- Adding this transition does not cause a contradiction.

Rules for when the second condition is satisfied:

- If s' has $X\varphi$ then t' must have φ.
- If s' does not have $X\varphi$ then t' must not have φ.
- If s' has $\varphi\, U\, \psi$ and does not have ψ then t' must have $\varphi\, U\, \psi$.
- If s has φ and does not have $\varphi\, U\, \psi$ then t' must not have $\varphi\, U\, \psi$.

Whiteboard: return to examples.

Pruning

The graph we get might not be a transition system at all, because some states might have no transitions out of them.

- Delete any state without a transition out of it.
- This might cause further deletions, as other states lose their transitions out.

The graph also might have the problem we observed when we worked with tableaux: endlessly delayed eventualities.

- Delete any state with $\varphi\, U\, \psi$ that has no path to ψ.
- Again, many deletions might ensue.

Completing the Model Check

Given a transition system $\mathcal{M}$ and LTL proposition φ, we have created a new, possibly bigger, transition system $\mathcal{M}'$, some of whose states might be labelled with φ.

The model check succeeds if all copies of the start state are labelled with φ.

We have not been quite precise enough about our algorithm to do a deep analysis of complexity, but there is no known algorithm with better than exponential time performance.

- Because the problem is PSPACE-complete, it is unlikely there ever will be.

Model Checking: Back to Practice

There is significant engineering and theoretical work needed to move from this set of slides to a technology capable of model checking a massive industrially relevant model.

But we can only do so much in one 2000/6000 level course!

In our first lecture we learned that logic started with philosophers. While we end with a logic of relevance to billions of dollars of risk management for giant commercial entities, it is worth remembering that even this logic began with philosophical questions, and required much mathematical work, before becoming computationally useful.

Who can say where the logics of the future will come from?