Model Checking LTL

Ralf Huuck

Australian Government
Department of Communications, Information Technology and the Arts
Australian Research Council

NICTA Members
ANU UNSW BusinessACT
NICTA Partners

Acknowledgments

Slides partially based on earlier lecture by Doron Peled.

Outline

1. Checking for invariants, deadlock etc.
2. Checking full LTL

1. Checking Invariants

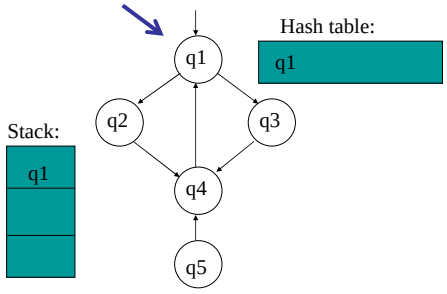
and more

Depth First Search

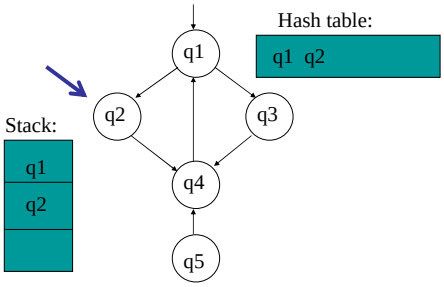
Program DFS
For each s such that Init(s)
    dfs(s)
end DFS

Procedure dfs(s)
for each s' such that
        R(s,s') do
    If new(s') then dfs(s')
end dfs.

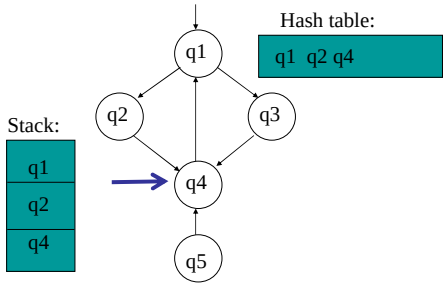
Start from an initial state



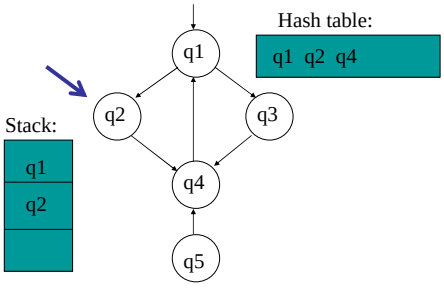
Continue with a successor



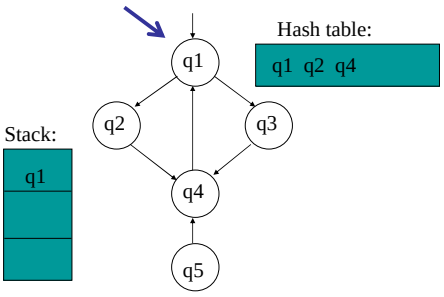
One successor of q2.



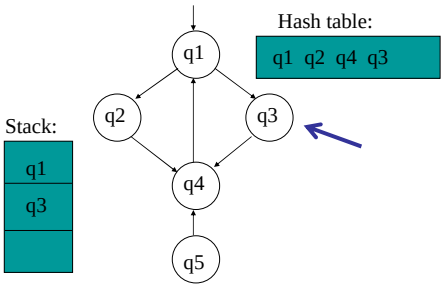
Backtrack to q2 (no new successors for q4).



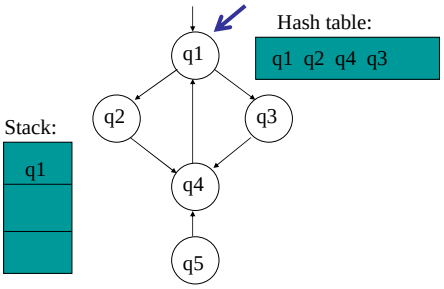
Backtracked to q1



Second successor to q1.



Backtrack again to q1.



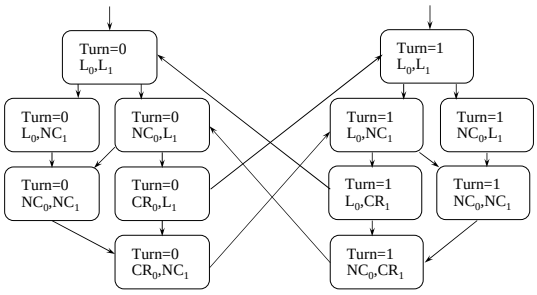
How can we check properties with DFS?

Invariants: check that all reachable states satisfy the invariant property. If not, show a path from an initial state to a bad state.

Deadlocks: check whether a state where no process can continue is reached.

Dead code: as you progress with the DFS, mark all the transitions that are executed at least once.

$G\neg(PC_0=CR_0 \wedge PC_1=CR_1)$ is in an invariant!



CR = critical region/section
NC= non-critical
Turn= turn variable in mutual exclusion algorithm

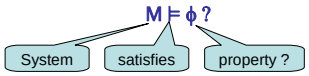
Want to do more!

- Want to check more properties.
- Want to have a uniform algorithm to deal with all kinds of properties.
- This is done by writing specification in temporal logic.
- Temporal logic specification can be translated into automata.

2. Checking LTL

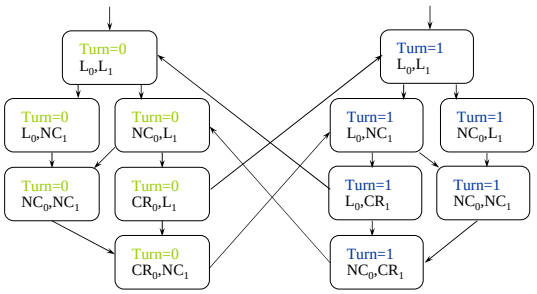
Model Checking

Model Checking Problem:

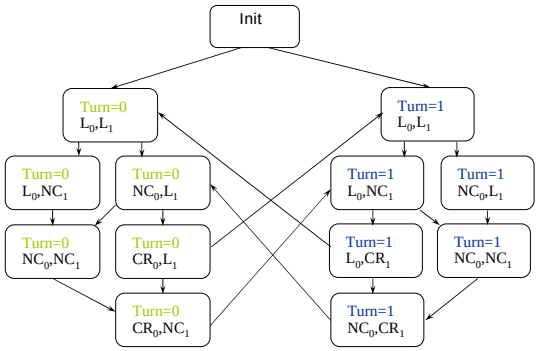


Typical: Model is given as Kripke structure/ω–Automata and property in temporal logic.

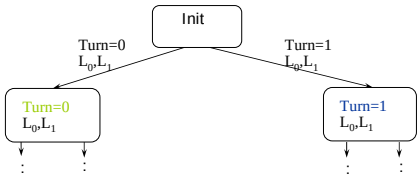
G (Turn=0 ⇒ F Turn=1)



Kripke Structure to Automaton



Kripke Structure to Automaton



- Add an additional initial node.
- Propositions are attached to incoming nodes.
- All nodes are accepting.

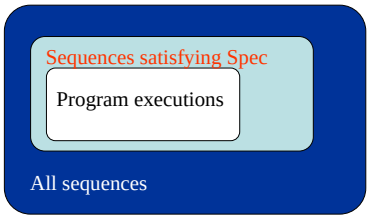
Model Checking as Inclusion Checking

- We want to find a correctness condition for a model to satisfy a temporal specification.
- Since both can be modeled as automata, we can check the relation between their languages.

- Language of a model: L(Model).
- Language of a specification: L(Spec).

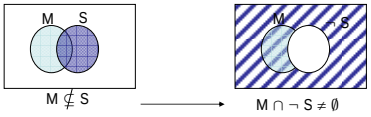
We need: $L(Model) \subseteq L(Spec)$.

Correctness



How to model-check?

- Show that $L(Model) \subseteq L(Spec)$.
- Equivalently: Complement
  Show that $L(Model) \cap \overline{L(Spec)} = \emptyset$.
- How? Check that $A_{model} \cap A_{\neg Spec}$ is empty.



What do we need to know?

$L(Model) \cap \overline{L(Spec)} = \emptyset$.

1. How to intersect two automata?
2. How to complement an automaton?
3. How to check for emptiness of an automaton ?
4. How to translate from LTL to an automaton?

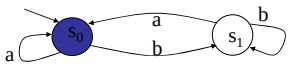
Intersecting Automata (re-visited)

1. Intersecting two automata (finite words)

- $A_1 = \langle \Sigma, S_1, \Delta_1, s_{01}, F_1 \rangle$ and $A_2 = \langle \Sigma, S_2, \Delta_2, s_{02}, F_2 \rangle$
- $A_1 \cap A_2$ =
  - Each state is a pair (s,t): $s \in S_1$ and $t \in S_1$.
  - Initial state: pair (s,t) such that $s=s_{01}$ and $t=s_{02}$.
  - Accepting states: pairs (s,t) such that $s \in F_1$ and $t \in F_2$
  - ((s,t) a (s',t')) is a transition if $(s,a,s') \in \Delta_1$, and $(t,a,t') \in \Delta_2$.

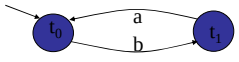
Example - intersecting two automata

A_1:



$L(A_1) = (a+b)^*a + \varepsilon$
(words ending with 'a' + empty word)

A_2:

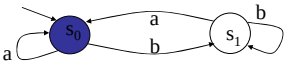


$L(A_2) = (ba)^* + (ba)^*b + \varepsilon$
(words that alternate between b and a + empty word)

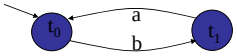
What should be the language of $A_1 \cap A_2$?

Example - intersecting two automata

A_1:



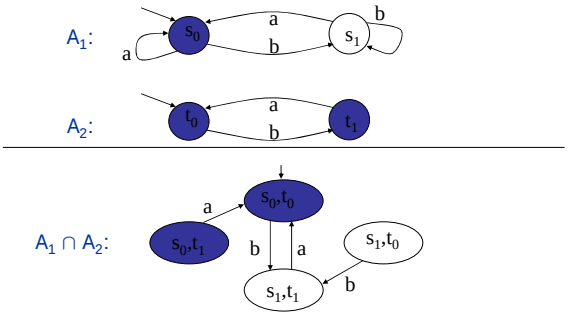
A_2:



$A_1 \cap A_2$:

1. States: (s_0,t_0), (s_0,t_1), (s_1,t_0), (s_1,t_1).
2. Initial state(s): (s_0,t_0).
3. Accepting states: (s_0,t_0), (s_0,t_1).

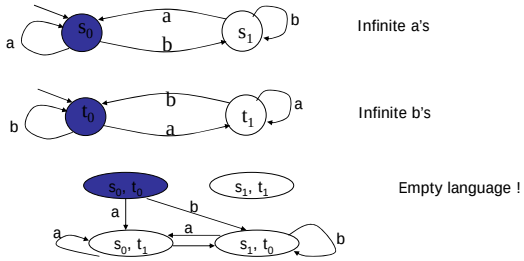
Example - intersecting two automata



$L(A_1 \cap A_2) = (ba)^* + \varepsilon$

Intersecting two Buchi automata (infinite words)

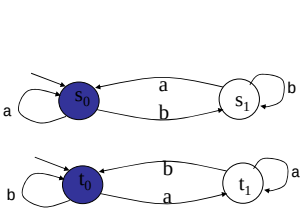
- Previous method doesn't work:



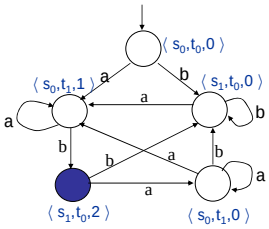
Intersecting two Buchi automata (infinite words)

- Strategy:
  - "Multiply" the product automaton by 3 (S = S1 × S2 × {0,1,2})
  - Start from the '0' copy.
  - Transition to the '1' copy when visiting a state from F1
  - Transition to the '2' copy if in a '1' state and visiting a state from F2, and in the next state back to a '0' state.
  - Make the '2' copy an accepting set.

Intersecting two Buchi automata (infinite words)



There are total of 12 states in the product automaton.
The reachable part of $A_1 \cap A_2$ is:



Complementing Automata

2. How to complement?

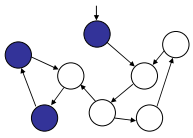
- Complementation is hard!
- If we know how to translate an LTL formula to a Buchi automaton, we can:
  1. Build an automaton A for φ, and complement A, **or**
  2. Negate the property, obtaining ¬φ (the sequences that should never occur). Build an automaton for ¬φ .

We will do 2., so we do not have to bother with complememtation.

Checking for Emptiness

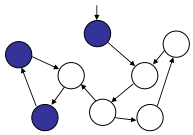
3. How to check for emptiness?

- Need to check if there exists an accepting run (passes through an accepting state infinitely often).
- This is called checking for emptiness, because if no such run exists, then L(A) = ∅



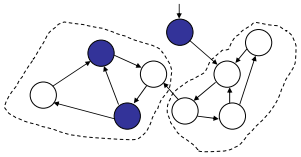
Emptiness and accepting runs

- If there is an accepting run, then it contains at least one accepting state an infinite # of times.
- This state must appear in a cycle.
- So, find a reachable accepting state (by DFS) on a cycle.
- How to detect cycles?



Finding accepting runs

- Rather than looking for cycles, look for SCCs:
  - A Strongly Connected Component (SCC): a set of nodes where each node is reachable from all others.
  - Finding SCC's is linear in the size of the graph.
  - Find a reachable SCC with an accepting node.

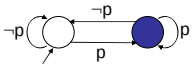


LTL to Automata

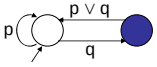
4. From LTL to automata

- Exponential blow-up
- Formulas are usually small

"always eventually p": GF p



"always p until q": G (pUq)



4. LTL to automata

Is there an algorithm to transform **LTL into Buchi**?

Yes , but

- transforming LTL into automata is non-trivial
- several approaches, none is obvious
- next lecture

Lessons learnt so far

- simple DFS algorithm allows to check for invariants, deadlocks etc.
- model checking problem can be seen as inclusion problem
- we had already all it takes, if only we knew how to translate LTL into Buchi