جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila
University of Mohamed Boudiaf - M'sila
1985

LTL model checking

Chapter 01: Introduction

Dr. Hichem Debbi

hichem.debbi@gmail.com

April 22, 2022

Model Checking

Model checking is an automatic formal method used for the verification of finite-state systems. Given a system model and such specification, which is a set of formal proprieties, the model checker verifies whether or not the model meets the specification. In case the specification is not satisfied, a counterexample is generated as an error trace.

Model Checking Founders

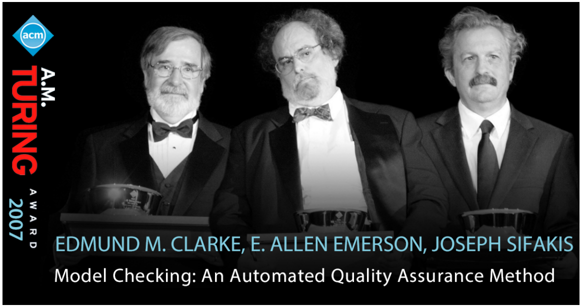

acm
A.M.
TURING
AWARD
2007
EDMUND M. CLARKE, E. ALLEN EMERSON, JOSEPH SIFAKIS
Model Checking: An Automated Quality Assurance Method

Figure: Model Checking Founders

Model checking process

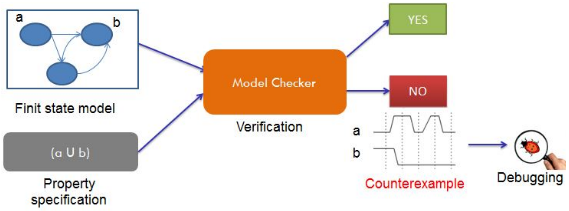


Figure: Model Checking

Model Checking Principles

- Systems are modeled by finite state machines
- Properties are written in temporal logic
- Verification algorithm is an exhaustive search of the state space
- Automatic generation of counterexamples

Any verification using model-based techniques is only as good as the model of the system.

Critical Examples

- A software controlling the trains collides
- A peace-maker cease to function
- A rocket my explode due to false data interpretation
- A malfunction of a vehicle's airbag

Modelling

Modelling: This task aims to deliver a model of the system using some model description language that can be accepted by a model checker. Despite the language used, it generally enables the representation of the system as finite-state automata, where states comprise information about the current values of variables and transitions describe how the system evolves from one state into another. In model checking, we refer to the transition system describing the behaviour of the system *Kripke structure* .

Transition systems

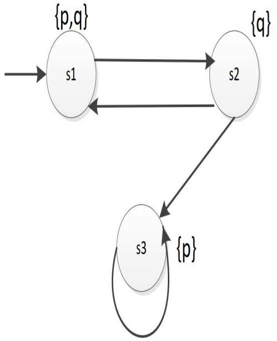
Transition systems

Transition systems represent directed graphs where the nodes represent the systems states, and the edges refer to state changes, or transitions. A state describes the current values of system variables at a certain moment. In a traffic light system, "red" for instance indicates the current color of the light.

Kripke strcture

A

Kripke structure is a variation of the transition system, originally proposed by Saul Kripke, used in model checking to represent the behavior of a system. It consists of a graph whose nodes represent the reachable states of the system and whose edges represent state transitions, together with a labelling function which maps each node to a set of properties that hold in the corresponding state. Temporal logics are traditionally interpreted in terms of Kripke structures.



(**Kripke Structure**) A Kripke structure is a tuple $M = (AP, S, S_0, R, L)$ that consists of a set AP of atomic propositions, a set S of states, $S_0 \in S$ an initial state, a total transition relation $R \subseteq S \times S$ and a labelling function $L : S \rightarrow 2^{AP}$ that labels each state with a set of atomic propositions.

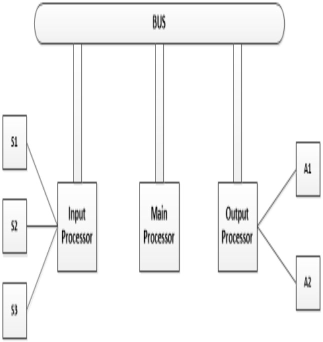
K description

$S = s1, s2, s3$. $S_0 = s1$.
$R = (s1, s2), (s2, s1)(s2, s3), (s3, s3)$.
$L = (s1, p, q), (s2, q), (s3, p)$.
On K we may produce a path
$\sigma = s1, s2, s1, s2, s3, s3, s3, \ldots$ and
$w = p, q, q, p, q, q, p, p, p,$ ... is the execution word over the path σ. K can produce execution words belonging to the language:

K description

$S = s1, s2, s3$. $S_0 = s1$.
$R = (s1, s2), (s2, s1)(s2, s3), (s3, s3)$.
$L = (s1, p, q), (s2, q), (s3, p)$.
On K we may produce a path
$\sigma = s1, s2, s1, s2, s3, s3, s3, ...$ and
$w = p, q, q, p, q, q, p, p, p,$ ... is the execution word over the path σ. K can produce execution words belonging to the language: $(p, qq) * (p)\omega \cup (p, qq)\omega$

Propositional Variables



- input processor is ok
- main processor fails
- bus has ready data to send
- actuator is failed

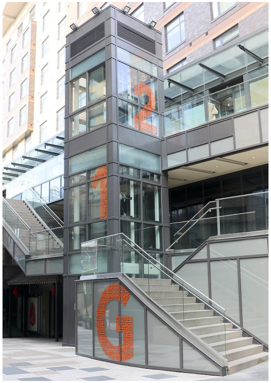
Specification

Before performing verification, a set of properties that should be satisfied by the model must be delivered, such as , a system should never crashes or it should always complete such task. This set of properties is given usually in a logical formalism, the mostly used is *temporal logic* since it is capable of representing how the system's behaviour evolves over time, where temporal logic formula is interpreted in the term of Kripke structure. Temporal logic is an extension of traditional propositional logic with modal operators. According to what we assume about time, temporal logics are either linear (Linear Temporal Logic (LTL)), or branching(tree) (Computation Tree Logic (CTL)). Using temporal logics we can express two main types of properties:

Specification examples

- Mutual exclusion:, only one user at a time can have access.
- **Finite time of usage**: a user can have access only for a finite amount of time.
- **Absence of individual starvation**: if a user wants to access to a resource, he/she eventually is able to do so.
- **Absence of blocking**: a user can always request to access a resource
- **Alternating access**: users must strictly alternate in using resources.

Properties



Safety & Liveness properties (Elevator control software can be model-checked to verify both *safety properties*, like "The cabin never moves with its door open", and *liveness properties*, like "Whenever the nth floor's call button is pressed, the cabin will eventually stop at the nth floor and open the door".

Why use temporal logic

- Requirements of concurrent, distributed, and reactive systems are often phrased as constraints on sequences of events or states or constraints on execution paths.
- Temporal logic provides a formal, expressive, and compact notation for realizing such requirements.
- The temporal logics we consider are also strongly tied to various computational frameworks (e.g., automata theory) which provides a foundation for building verification tools.

Büchi Automaton

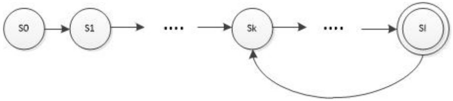
Definition

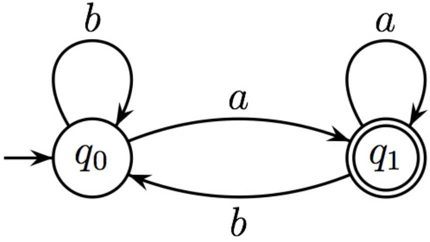
(**Büchi Automaton**) A Büchi automaton is a tuple $B = (Q, Q_0, E, \Sigma, F)$ where Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $E \subseteq Q \times Q$ is the transition relation,Σ is a finite alphabet, and $F \subseteq Q$ is the set of accepting or final states.

We use Büchi automaton to define a set of infinite words of an alphabet. A path is a sequence of states $(q_0 q_1 ..., q_k)$, $k \geq 1$ such that $(q_i, q_{i+1}) \in E$ for all $1 \leq i < k$. A path $(q_0 q_1 ..., q_k)$ is a cycle if $q_k = q_1$, the cycle is accepting if it contains a state in F.

Büchi Automaton

A path $(s_0 s_1 ..., s_k ....s_l)$ where $l > k$ is an accepting if $s_k ...s_l$ forms an accepting cycle. We call a path that starts at the initial state and reaches an accepting cycle an accepting path or counterexample. A minimal counterexample is an accepting path with minimal number of transitions.

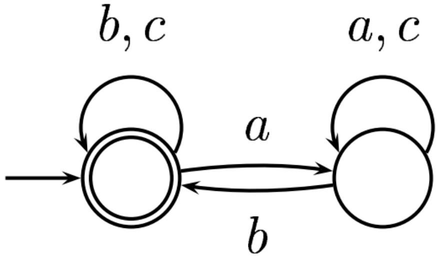




A Büchi automaton with two states, q_0 and q_1, the former of which is the start state and the latter of which is accepting. Its inputs are infinite words over the symbols $\{a, b\}$. As an example, it accepts the infinite word $(ab)^\omega$, where x^ω denotes the infinite repetition of a string. It rejects the infinite word $aaab^\omega$.

Accepting runs

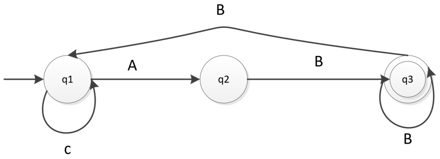
- A run $q0, q1, q1, q1, q0$ on *aacb* is accepting
- A run $q0, q0, q0, q0, q0$ on *bbbb* is accepting
- A run $q0, q0, q1, q1, q1$ on *baac* is rejecting



The language is $\epsilon, b, bb, ccc, bab, \ldots$ That is, a regular expression: $\epsilon + a(a+c) * b(b+c)$

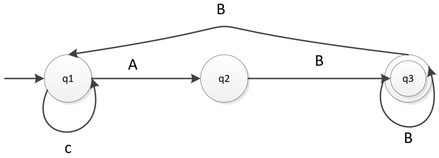
Accepting runs

Consider the automaton with the alphabet $\Sigma = \{A, B, C\}$



- the word C^ω —> one run $q1q1q1$ or $q1^\omega$
- the word AB^ω —> $q1q2q3^\omega$
- the word $(CABB)^\omega$ —> $(q1q1q2q3)^\omega$
- the word $(ABB)^nC^\omega$ —> $(q1q2q3)^nq1^\omega$ where $n \geqslant 0$

Accepting runs

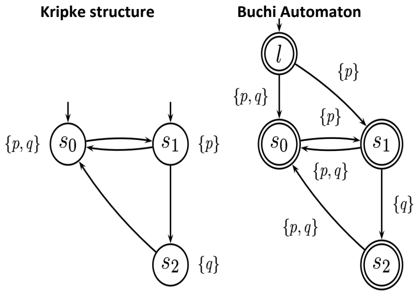


- Runs that go infinately often through the accept state $q3$ are accepting
- $q1q2q3^{\omega}$ and $(q1q1q2q3)^{\omega}$
- $q1^{\omega}$ is not accepting because it never visits the accepting state $q3$
- paths of the form $(q1q2q3)^{n}q1^{\omega}$ are not accepting because they visit $q3$ only finitely many times.

The language accepted by this automaton is given by the regular $\omega-$regular expression: $C^{*}AB(B^{+}+BC^{*}AB)^{\omega}$

Modeling Systems Using Automata

Every system is represented by a set of its executions.
Thus, every state is accepting.
Transform the Kripke structure into Büchi automaton



Modeling Systems Using Automata

Transform Kripke structure (S, R, S_0, L), where $L : S \rightarrow 2^{AP}$ into automaton

- where $\Sigma = 2^{AP}$
- $(I, a, s') \in E$ iff $s \in S_0$ and $a = L(s)$
- $(s, a, s) \in E$ iff $(s, s') \in R$ and $a = L(s')$

So a transition gets the label of the source state.

Temporal logic

Temporal logic

Temporal logic is an extension of traditional propositional logic with modal operators. According to what we assume about time, temporal logics are either linear (Linear Temporal Logic (LTL)), or branching(tree) (Computation Tree Logic (CTL)).

Temporal modalities

the temporal modalities of LTL allow us to formalize properties that involve time and encode formulae about the future on paths.

LTL and Buchi Automata

Buchi automata can encode all LTL properties.

LTL Syntax

The syntax of LTL state formula over the set *AP* are given as follows :

$$\varphi ::= true|a|\neg\phi|\phi_1 \wedge \phi_2|\bigcirc\phi|\phi_1 \cup \phi_2$$

$a \in AP$ is an atomic proposition, $\bigcirc(X)$ is the next operator, and *U* is the until operator. The other Boolean connectives can be simply derived using the Boolean connectives $\neg$ and $\wedge$. The *eventual* operator *F* and the *always* operator and *G* can be easily derived using the temporal operator *U*.

Semantics

Given a path $\pi = s_0 s_1 ...$ and an integer $j \geq 0$, where $\pi[j] = s_j$, such that $Words(\varphi) = \{\pi \in (2^{AP})^w)\sigma \models \varphi\}$, the semantics of LTL formulas for infinite words over 2^{AP} are given as follows:

$\pi \models true \Leftrightarrow true$
$\pi \models a \Leftrightarrow a \in L(s_0)$
$\pi \models \neg\varphi \Leftrightarrow \pi \not\models \varphi$
$\pi \models \varphi_1 \wedge \varphi_2 \Leftrightarrow s \models \varphi_1 \wedge s \models \varphi_2$
$\pi \models X\varphi \Leftrightarrow \pi\,[1..] \models \varphi$
$\pi \models \varphi_1 \mathbf{U} \varphi_2 \Leftrightarrow \exists j \geq 0 . \pi\,[j..] \models \varphi_2 \wedge (\forall 0 \leq k < j . \pi\,[k..] \models \varphi_1)$

The semantics for the derived operators F and G is given as follows :

$\pi \models F\varphi \Leftrightarrow \exists j \geq 0 . \pi\,[j..] \models \varphi$
$\pi \models G\varphi \Leftrightarrow \forall j \geq 0 . \pi\,[j..] \models \varphi$

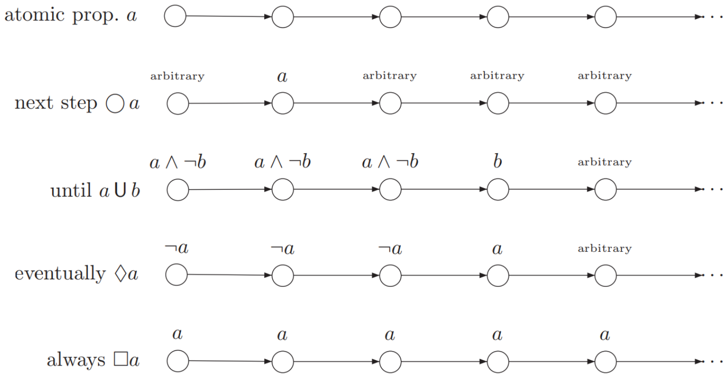
Alternative notations

$X \rightarrow \bigcirc$

$F \rightarrow \Diamond$

$G \rightarrow \Box$

Basic semantics of temporal modalities



Basic semantics of temporal modalities

- Both operators $\Box$ and $\Diamond$ are duals:
  $\neg\Box\varphi \equiv \Diamond\neg\varphi$
- where $\Diamond$ can be rewritten in terms of until U:
  $\Diamond\varphi \equiv \mathit{true} \cup \varphi$

LTL Examples

- $\Box\neg(read \wedge write)$ In all future states, it is not possible to have read and write
- $\Box(request \implies \Diamond grant)$ At every a future state, a request implies that there exists a future state where grant holds.
- $\Box(request \implies (request \cup grant))$ At every position in the future, a request implies that there exists a future point where grant holds, and request holds up until that point.

LTL model checking

Verifying whether a finite state system described in Kripke structure A_M satisfies an LTL property φ reduces to the verification that $A = A_M \cap A_{\neg\varphi}$ has no accepting path, where $A_{\neg\varphi}$ refers to the Büchi automaton that violates φ, $L_\omega(A) = Words(\neg\varphi)$. We call this procedure a test of emptiness. So, In case $A_M \cap A_{\neg\varphi} \neq \emptyset$ a counterexample is generated.

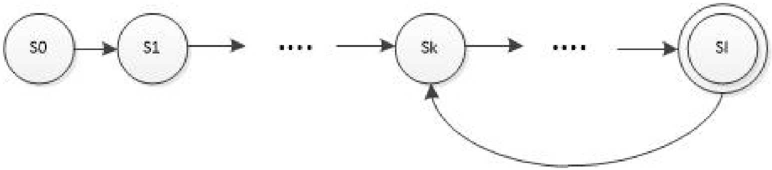
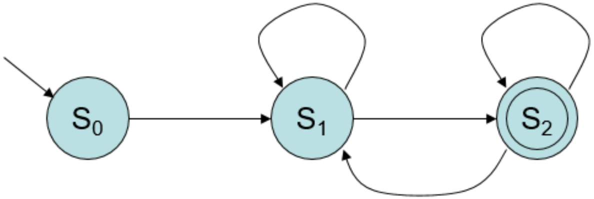
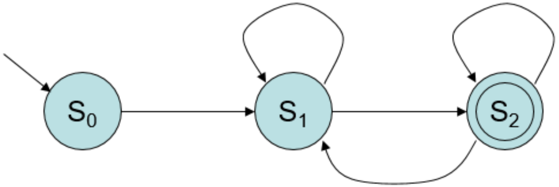


Figure: Accepting path (Counterexample)

Accepting runs – Example



Accepting runs – Example



$$\begin{tikzcd} \arrow[r] & S_0 \arrow[r] & S_1 \arrow[loop above] \arrow[r] & S_2 \arrow[loop above] \arrow[l, bend left] \end{tikzcd}$$

$\sigma_1 = S_0 S_1 S_2 S_2 S_2 S_2 \ldots$ ACCEPTED

$\sigma_2 = S_0 S_1 S_2 S_1 S_2 S_1 \ldots$ ACCEPTED

$\sigma_3 = S_0 S_1 S_2 S_1 S_1 S_1 \ldots$ REJECTED

LTL model checking– Algorithm

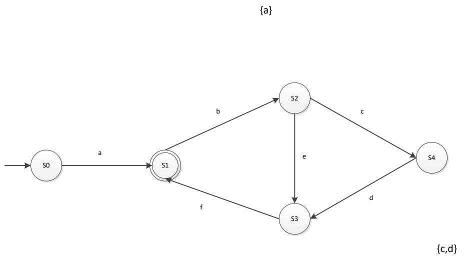
Nested Depth First Search

- How do we find accepted sequences?
- Accepted sequences must contain a cycle
- In order to contain accepting states infinitely often We are interested only in cycles that contain at least an accepting state
- During depth first search start a second search when we are in an accepting states If we can reach the same state again we have a cycle (and a counterexample)

LTL model checking– Algorithm

Emptiness of Buchi Automata

An automation is non-empty iff: there exists a path to a cycle containing an accepting state

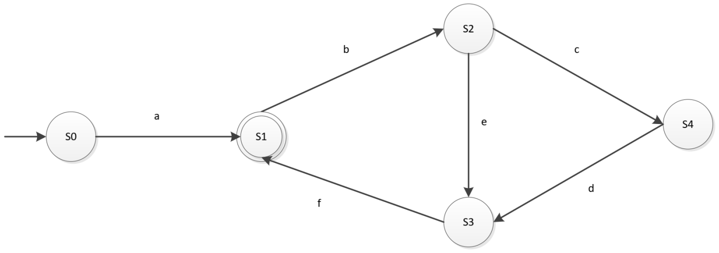


LTL model checking– Algorithm

is this automaton empty?

LTL model checking– Algorithm

is this automaton empty? No – it accepts $a(bef)^\omega$


$$\begin{tikzcd}
 & & S2 \arrow[dr, "c"] \arrow[d, "e"] & \\
\arrow[r] & S0 \arrow[r, "a"] & S1 \arrow[u, "b"] & S4 \arrow[dl, "d"] \\
 & & S3 \arrow[ul, "f"] &
\end{tikzcd}$$

Product of two Buchi automaton

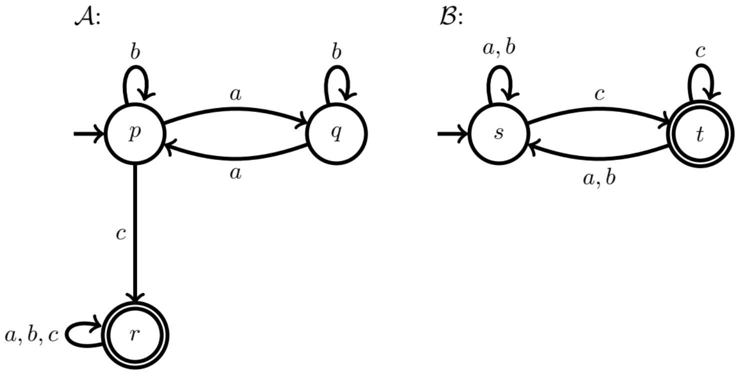
Let $A_1 = (Q_1, Q_0^1, E_1, \Sigma, F_1)$ and $A_2 = (Q_2, Q_0^2, E_2, \Sigma, F_2)$
Then, $A_1 X A_2 = (Q, Q_0, E, \Sigma, F)$ where:

- $Q = Q_1 \times Q_2 \times \{A_1, A_2\}$
- $Q_0 = Q_0^1 \times Q_0^2 \times A_1$
- $F = F1xF2 \times A_1$
- $< p, q > \xrightarrow{a} < p', q' >$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$

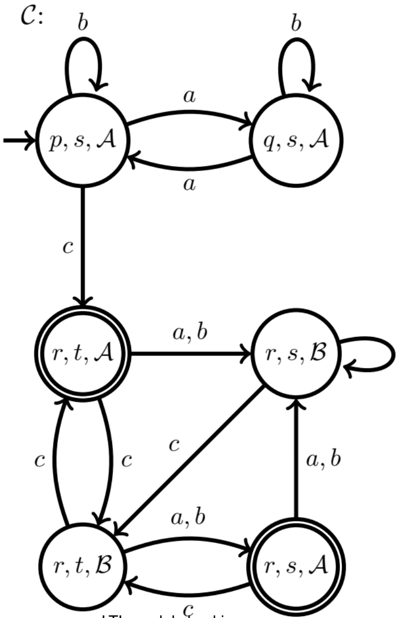
Product of two Buchi automaton

- $< p, q, A_1 > \xrightarrow{a} < p', q', A_1 >$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $p \notin F_1$
- $< p, q, A_1 > \xrightarrow{a} < p', q', A_2 >$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $p \in F_1$
- $< p, q, A_2 > \xrightarrow{a} < p', q', A_2 >$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $q \notin F_2$
- $< p, q, A_2 > \xrightarrow{a} < p', q', A_1 >$ iff $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $q \in F_2$

Product of two Buchi automaton - Example



Product of two Buchi automaton - Example



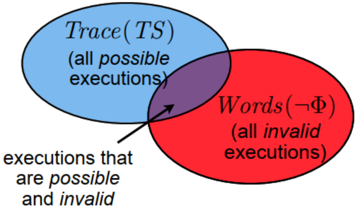
Buchi automaton & LTL formulae

Given:

- Transition system TS
- LTL formula Φ

Question: Does TS satisfy Φ, i.e.,

$$TS \models \Phi \; ?$$


$Trace(TS)$
(all *possible* executions)
$Words(\neg\Phi)$
(all *invalid* executions)
executions that are *possible* and *invalid*

Answer (conceptual):

$$TS \models \Phi$$ [*TS* satisfies Φ]

$$\Updownarrow$$

$$Trace(TS) \subseteq Words(\Phi)$$ [All executions of *TS* satisfy Φ]

$$\Updownarrow$$

$$Trace(TS) \cap Words(\neg\Phi) = \emptyset$$ [No execution of *TS* violates Φ]

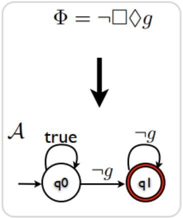
How to determine whether $Trace(TS) \cap Words(\neg\Phi) = \emptyset$ **?**

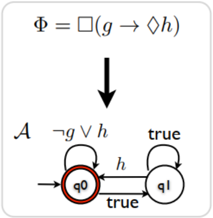
?

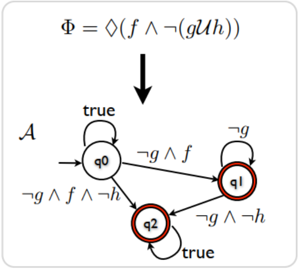
Buchi automaton & LTL formulae

Theorem. *There exists an algorithm that takes an LTL formula Φ and returns a Büchi automaton $\mathcal{A}$ such that*

$$Words(\Phi) = \mathcal{L}_\omega(\mathcal{A})$$






$\Phi = \Diamond(f \wedge \neg(g\mathcal{U}h))$
$\mathcal{A}$
true
$\neg g$
q0
$\neg g \wedge f$
q1
$\neg g \wedge f \wedge \neg h$
q2
$\neg g \wedge \neg h$
true

Buchi automaton & LTL formulae

Given:

- Transition system TS
- LTL formula Φ
- NBA $\mathcal{A}_{\neg\Phi}$ accepting $\neg\Phi$ with the set *F* of accepting states

$$TS \not\models \Phi$$

$$\Updownarrow$$

$$Trace(TS) \not\subseteq Words(\Phi)$$

$$\Updownarrow$$

$$Trace(TS) \cap Words(\neg\Phi) \neq \emptyset$$

$$\Updownarrow$$

$$Trace(TS) \cap \mathcal{L}_\omega(\mathcal{A}_{\neg\Phi}) \neq \emptyset$$

$$\Updownarrow$$

$$TS \otimes \mathcal{A}_{\neg\Phi} \not\models \text{“eventually forever”} \neg F$$

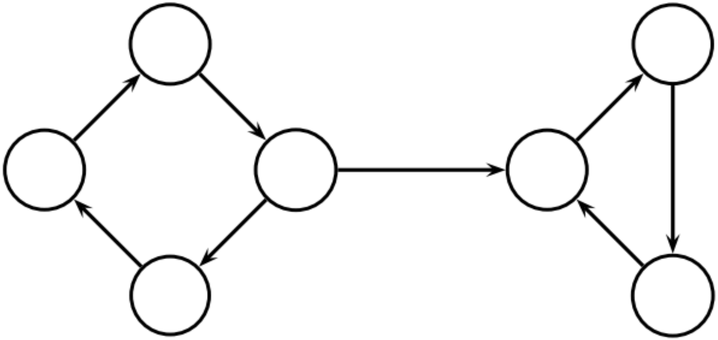
LTL model checking

- LTL Model-Checking = Emptiness of Büchi automata
- Algorithm: depth-first search (DFS) on graph
- **Problem**:the graph is **Huge** = state-explosion

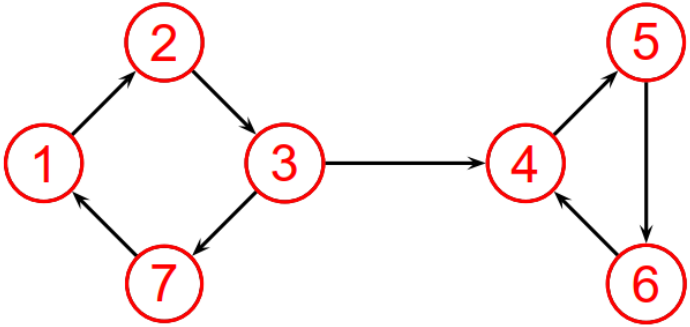
1985

جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila

LTL model checking– DFS



LTL model checking– DFS



جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila

State explosion problem

State-of-the-art model checkers can handle state spaces of about 10^8 to 10^9 states with explicit state-space enumeration. Using clever algorithms and tailored data structures, larger state spaces (10^{20} up to even 10^{476} states) can be handled for specific problems.

Safety properties

Safety: nothing bad ever happens

- Invariants: "x is always less than 10"
- Deadlock freedom: "the system never reaches a state where no moves are possible"
- Mutual exclusion: "the system never reaches a state where two processes are in the critical section" As soon as you see the "bad thing", you know the property is false (so they are falsified by a finite prefix of an execution trace)

Liveness properties

Liveness: something good will eventually happen

- Termination: "the system eventually terminates"
- Response: "if action X occurs then eventually action Y will occur" Need to keep looking for the ”good thing” forever Liveness can only be falsified by an infinite-suffix of an execution trace
- Practically, a counter-example here is a set of states starting from initial one and going through a loop where you get stuck and never reach the "good thing".

References

- Christel Baier and Joost-Pieter Katoen. Principles of model checking. MIT Press. 2008.
- Arie Gurfinkel. Automata-Theoretic LTL Model-Checking