Systems Verification

Alessandro Abate

Day 1
January 25, 2016
Course setup

Intro to formal verification

Models - labelled transition systems

Properties as specifications - modal logics

Model checking algorithms
Instructors, TAs, times

- Alessandro Abate (day 1, 4)
- Marta Kwiatkovska (day 2, 3)
- Daniel Kroening (day 5)

- Maria Svorenova, Sadegh E.Z. Soudjani

- Lectures in the mornings
- Practicals in the afternoons (except for Friday)
Organisation of the module

- assessment: participation to lectures and practicals
- readings
- tools
  - SPIN/nuSMV, PRISM, FAUST2
Objectives of the course

- To familiarise with the main concepts in systems modelling and property specification
- To explain the fundamentals of explicit algorithms in qualitative and quantitative model checking
- To gain practical experience of verification tools and how they are applied through examples
- To give appreciation of relevant research directions in systems verification and synthesis
High-level overview of the course

- Formalise models of systems under verification
- Formally study properties (verify specifications) of models expressed in some logics
- Develop algorithms and data structures to check that the model of a system satisfies a given property
- Applications to hardware and software systems, extensions to more general systems (biology, autonomy, robotics)
List of topics to be covered in the course

**day 1** Basics of verification. Transition systems, quantitative requirements, temporal logics (LTL and CTL). Fundamentals of algorithmic verification (via model checking).

**day 2** Verification of Markov models, time and rewards, probabilistic temporal logics. Overview of algorithms, including statistical model checking.

**day 3** Verification for models with non-determinism/decisions: Markov decision processes, issues of time and rewards. Stochastic games. Main algorithms for verification and strategy/controller synthesis.

**day 4** Hybrid systems. From discrete to continuous models. The role of stochasticity. Formal abstractions for automated verification and symbolic controller synthesis. Software tools for verification and synthesis of complex control models.

**day 5** Propositional satisfiability (SAT) checking, SMT solving. Bounded model checking (BMC). Unbounded verification with SAT/SMT and inductive reasoning. Applications to software.
Outline

Course setup

Intro to formal verification

Models - labelled transition systems

Properties as specifications - modal logics

Model checking algorithms
What is verification?

“Have you actually made what you were trying to make?”

- verification vs validation
  1. Does the hardware/software system satisfy all the properties characterising its specification?
  2. Is the specification of the system correct? (“Have you made the right thing?”)

  1. a-priori, provably correct-by-design
  2. a-posteriori, tested to work correctly
Limitations of validation

- validation by testing
  - manual inspection, very costly and error prone
- validation via simulation
  - computer-based, but often lengthy and imprecise
Limitations of validation

- validation by testing
  - manual inspection, very costly and error prone
- validation via simulation
  - computer-based, but often lengthy and imprecise

“Testing/simulation shows the presence of errors, does not prove their absence” [Edsger W. Dijkstra]
Bugs/errors are everywhere (particularly where they shouldn’t be)

- Therac-25 patients receiving excessive radiation;
- Bank of New York corrupted database of bond market;
- FPU of Pentium errors;
- Ariane explosion;
- Knight Capital Group default, Denver Airport halt, Toyota Prius recall...
The practical need for formal verification

Not only reasons of safety, it's also about the money . . .

- Bank of New York bug: $ 5 million
- Pentium bug: $ 475 million
- Ariane bug: $ 500 million
- Blackout bug: $ 6 billion
- Denver bug: $ 300 million
- Knight bug: $ 440 million

“The cost of software bugs to the U.S. economy is estimated at $ 60 billion per year.”

National Institute of Standards and Technology, 2002
When are bugs introduced and detected?

About 50% of all defects are introduced during programming, the phase in which actual coding takes place. Whereas just 15% of all errors are detected in the initial design stages, most errors are found during testing. At the start of unit testing, which is oriented to discovering defects in the individual software modules that make up the system, a defect density of about 20 defects per 1000 lines of (uncommented) code is typical. This has been reduced to about 6 defects per 1000 code lines at the start of system testing, where a collection of such modules that constitute a product are tested. On launch of a new software release, the typical accepted software defect density is about one defect per 1000 lines of code.

Errors are typically concentrated in a few software modules – about half of the modules are defect free, and about 80% of the defects arise in a small fraction (about 20%) of the modules – and often occur when interfacing modules. The repair of errors that are detected prior to testing can be done rather economically. The repair cost significantly increases from about $1000 (per error repair) in unit testing to a maximum of about $12,500 when the defects are demonstrated during system operation. It is of vital importance to seek techniques that find defects as early as possible in the software design process: the costs to repair them are substantially lower, and their influence on the rest of the design is less substantial.

Hardware Verification

Preventing errors in hardware design is vital. Hardware is subject to high fabrication costs; fixing defects after delivery to customers is difficult, and quality expectations are high. Whereas software defects can be repaired by providing patches, for some products this is much higher, though. Microsoft has acknowledged that Windows 95 contained at least 5000 defects. Despite the fact that users were daily confronted with anomalous behavior, Windows 95 was very successful.

Why are bugs introduced?

Hardware and software systems - particularly when embedded within physical components - are among the most complex artefacts ever produced by humans.

Pentium 4 microprocessor

- transistors: 55 million
- area: 146 mm²
- clock rate: $3.2 \times 10^9$ Hz
How does verification detect bugs?

- property checking: does system (model) $S$ satisfy property $P$?
- equivalence checking: are systems (models) $S_1$ and $S_2$ equivalent?

- in this course, the focus will be on property checking
What is *formal* about formal verification?

- the system is *formalised* as some mathematical model
- the property is *formalised* within a temporal logic
- checking whether the system satisfies the property is *formalised*, that is formally proven
Why is formal verification *computer-aided*?

- a formal proof that a system satisfies a property may consist of millions of steps
- without the use of clever algorithms and data structures, proving that a system satisfies a property can be quite impractical
- automatic techniques are user- and industry-friendly
- formal verification can be performed via *model checking*
Model checking: a computer-aided, formal verification paradigm

model checker:

does $M$ verify $p$?

yes

no: counterexample
The industrial impact of model checking

- verification of protocols, circuits, hardware, software
Outline

Course setup

Intro to formal verification

Models - labelled transition systems

Properties as specifications - modal logics

Model checking algorithms
Model of a system

- given a system (e.g., a physical device, a hardware circuit, or software code), a model represents an abstraction of it
- *all models are wrong, but some are useful* [G.E.P. Box]
- many levels of abstraction are possible → many models of a system can be made; abstraction/refinement (choice of right model) ought to be part of the modelling effort
- formal verification asserts properties of a *model*, not of the underlying *system*
(Labeled) transition system

Definition
A transition system is a tuple \( \langle S, \rightarrow, I, AP, L \rangle \) consisting of
- a set \( S \) of states,
- a transition relation \( \rightarrow \subseteq S \times S \),
- a set \( I \subseteq S \) of initial states,
- a set \( AP \) of atomic propositions (alphabet), and
- a labelling function \( L : S \rightarrow 2^{AP} \).

- a TS is also known as Kripke structure [CGP99]
- internal non-determinism
- a TS can be as well action enabled [BK08] (next slide)
Definition
A transition system is a tuple \( \langle S, \text{Act}, \rightarrow, I, \text{AP}, L \rangle \) consisting of
- a set \( S \) of states,
- a set \( \text{Act} \) of actions,
- a transition relation \( \rightarrow \subseteq S \times \text{Act} \times S \),
- a set \( I \subseteq S \) of initial states,
- a set \( \text{AP} \) of atomic propositions (alphabet), and
- a labelling function \( L : S \rightarrow 2^{\text{AP}} \).

- action enabled TS are closely related to Moore machines
- actions: internal vs. external non-determinism
- henceforth, for simplicity we will work with the first definition
On LTS models

- can write $s \rightarrow s'$ instead of $(s, s') \in \rightarrow$

- we consider exclusively transition relations such that each state has an outgoing transition, that is

$$\forall s \in S : \exists s' \in S : s \rightarrow s'$$

this is known as *non-blocking* condition, as absence of a *terminal state*

- here we consider *finite TS*, namely assume $S$ has finite cardinality
Example: model of a traffic light

System description: A traffic light can be red, green, amber or black (not working). The traffic light might stop working at any time. After it has been repaired, it turns red. Initially, the light is red.

\[ S = \{1, 2, 3, 4, 5\} \]

1. red
2. amber and red
3. green
4. amber
5. black
**Example: model of a traffic light**

**System description:** A traffic light can be red, green, amber or black (not working). The traffic light might stop working at any time. After it has been repaired, it turns red. Initially, the light is red.

\[
\begin{align*}
S &= \{1, 2, 3, 4, 5\} \\
\rightarrow &= \{(1, 2), (2, 3), (3, 4), (4, 1), (1, 5), (2, 5), (3, 5), (4, 5), (5, 1)\} \\
I &= \{1\} \\
AP &= \{r, a, g\} \\
L &= \{1 \mapsto \{r\}, 2 \mapsto \{r, a\}, 3 \mapsto \{g\}, 4 \mapsto \{a\}, 5 \mapsto \emptyset\}
\end{align*}
\]
Example: model of a traffic light

**System description:** A traffic light can be red, green, amber or black (not working). The traffic light might stop working at any time. After it has been repaired, it turns red. Initially, the light is red.
Direct predecessors and successors

**Definition**

For \( s \in S \), the set \( \text{Post}(s) \) of *direct successors* of \( s \) is defined by

\[
\text{Post}(s) = \{ s' \in S \mid s \rightarrow s' \}\.
\]

For \( s \in S \), the set \( \text{Pre}(s) \) of *direct predecessors* of \( s \) is defined by

\[
\text{Pre}(s) = \{ s' \in S \mid s' \rightarrow s \}\.
\]
**Post**\(^*\) and **Pre**\(^*\)

**Definition**

The set \(Post^*(s)\) consists of the states reachable in the state graph from \(s\).

Let \(C \subseteq S\). The set \(Post^*(C)\) is defined by

\[
Post^*(C) = \bigcup_{s \in C} Post^*(s).
\]

The set \(Pre^*(s)\) consists of the states that can reach \(s\) in the state graph.

Let \(C \subseteq S\). The set \(Pre^*(C)\) is defined by

\[
Pre^*(C) = \bigcup_{s \in C} Pre^*(s).
\]

- \(Reach(TS) = Post^*(I)\) is the reachability set of TS from \(I\)
- related to safety analysis (and used for CTL model checking)
Determinism of a transition system

Definition
A TS $\langle S, \rightarrow, I, AP, L \rangle$ is deterministic if $|I| = 1$, and if $\forall s \in S, |\text{Post}(s)| = 1$, where $|\cdot|$ denotes the cardinality of a set. Else the TS is non-deterministic.

(above notion refers to internal non-determinism)
Path fragment, path, trace

- nomenclature: *path, run, execution, trajectory* are synonyms

Definition
Consider the transition system $\langle S, \rightarrow, I, AP, L \rangle$.

A **finite path fragment** is a finite state sequence $s_0s_1\ldots s_n$ for some $n \geq 0$ such that $s_i \in Post(s_{i-1})$ for all $0 < i \leq n$.

An **infinite path fragment** is an infinite state sequence $s_0s_1\ldots$ such that $s_i \in Post(s_{i-1})$ for all $i > 0$.

A path fragment is **initial** if $s_0 \in I$.

A **path** is an initial infinite path fragment $\rightarrow Paths(TS)$

A **trace** is the “output” of a path: $L(s_0)L(s_1)\ldots$
Give an example of a finite initial path fragment:

Give an example of an initial infinite path fragment (a path):
Give an example of a finite initial path fragment: 12341
Give an example of an initial infinite path fragment (a path): (15)ω
Some notational conventions

- Let $\pi = s_0s_1 \ldots$ be an infinite path fragment (notions below apply to finite path as well).

- For $j \geq 0$, the $j$th state of $\pi$, $s_j$ is denoted by $\pi[j]$ (initial state is indexed by 0).

- For $j \geq 0$, the $j$th prefix of $\pi$, $s_0s_1 \ldots s_j$ is denoted by $\pi[..j]$

  $$\pi[..j] = s_0s_1 \ldots s_{j-1}s_j s_{j+1} \ldots$$

- For $j \geq 0$, the $j$th suffix of $\pi$, $s_js_{j+1} \ldots$ is denoted by $\pi[j..]$

  $$\pi[j..] = s_0s_1 \ldots s_{j-1}s_j s_{j+1} \ldots$$

- The set of infinite path fragments $\pi$ with $\pi[0] = s$ is denoted by $Paths(s)$.
Some notational conventions (example)

Let $\pi = 1234512$

What is $\pi[4]$?

What is $\pi[..3]$?

What is $\pi[5..]$?
Let $\pi = 1234512$

What is $\pi[4]$? 5

What is $\pi[..3]$? 1234

What is $\pi[5..]$? 12
What is $\text{Paths}(5)$?

What is the set of paths of the transition system?
What is $Paths(5)$?  \{$(51)^\omega, (51)^* \ 5 \ Paths(1)$\}

What is the set of paths of the transition system?  $Paths(TS) = Paths(1) = \{(1234)^\omega, (1234)^* \{1, 12, 123\} Paths(5) = \{((1234)^*(1\{\epsilon, 2, 23, 234\}5)^*)*\{1234, 1\{\epsilon, 2, 23, 234\}5\}^\omega\}$
Example 2: deterministic, finite transition system

- Consider system with 2 variables \((x, y)\), ranging over \(\{0, 1\}\)
- System starts in state \((1, 1)\), and consists of transition
  \[ x := (x + y) \mod 2 \]

- TS can be built as
  - \(S = \{0, 1\} \times \{0, 1\}\)
  - \(I = \{(1, 1)\}\)
  - \(\rightarrow = \{(1, 1), (0, 1), (0, 1), (1, 1), (1, 0), (1, 0), (0, 0), (0, 0)\}\)
  - \(2^{AP} = S\), \(L\) is the identity map
Example 2: deterministic, finite transition system

- Consider system with 2 variables \((x, y)\), ranging over \(\{0, 1\}\).
- System starts in state \((1, 1)\), and consists of transition \(x := (x + y) \mod 2\).

TS can be built as

- \(S = \{0, 1\} \times \{0, 1\}\)
- \(I = \{(1, 1)\}\)
- \(\rightarrow = \{(1, 1), (0, 1), (0, 1), (1, 1), (1, 0), (1, 0), (0, 0), (0, 0)\}\)
- \(2^{AP} = S\), \(L\) is the identity map

TS path, for the given \(I\), is unique.
Example 3: transition system for an ODE

- Ordinary difference equation (ODE)

\[ x(k + 1) = f(x(k)), \quad x(0) = x_0, \quad y(k) = Cx(k) \]

- e.g. \( f(x(k)) = Ax(k); \ k \in \mathbb{N}, \ x \in \mathbb{R}^n \)
Example 3: transition system for an ODE

- Ordinary difference equation (ODE)

\[ x(k + 1) = f(x(k)), \quad x(0) = x_0, \quad y(k) = Cx(k) \]

- e.g. \( f(x(k)) = Ax(k); \quad k \in \mathbb{N}, \quad x \in \mathbb{R}^n \)

- consider TS where
  - \( S = \mathbb{R}^n \)
  - \( x \rightarrow x' \iff x' = f(x) = Ax \)
  - \( I = x_0 \)
  - \( 2^{AP} = \mathbb{R}^n \), \( L \) is a linear function with scaling factor \( C \)

- TS is deterministic, but (uncountably) infinite (and with continuous labels set) – more about this model on Thursday
Example 4: Timed automaton

[
\dot{c} = 1 \\
2 \leq c \leq 10: \text{reset}(c) \\
1 \leq c \leq 2: \text{reset}(c) \\
1 \leq c \leq 10: \text{reset}(c) \\
]

(not covered in this course)
Example 5: Hybrid automaton

Variable $c$ is a vector

(covered in this course on day 4)
Example 6: Markov chain

\[ P = \begin{bmatrix}
0 & 0.99 & 0 & 0 & 0 & 0.01 \\
0 & 0 & 0.95 & 0 & 0 & 0.05 \\
0 & 0 & 0 & 0.99 & 0 & 0.01 \\
0.99 & 0 & 0 & 0 & 0 & 0.01 \\
0.9 & 0 & 0 & 0 & 0 & 0.1 \\
\end{bmatrix} \]

- (covered in this course on day 2)
- Markov decision process: M. chains “with actions” (day 3)
Example x: Stochastic hybrid models

- stochastic differential equations as special instances
- to be discussed on day 4
Course setup

Intro to formal verification

Models - labelled transition systems

Properties as specifications - modal logics

Model checking algorithms
Modal logics

- based on propositional and predicate logic
- used to reason about objects with modalities (expressed via modal operators)
- in particular, modal operators qualify temporal expressions
- in this course we shall focus on two classes: LTL and CTL
  1. LTL: linear temporal logic
  2. CTL: computational tree logic
- allows encoding fine-grained, complex statements in natural language
Syntax of LTL

▶ \( \varphi ::= \text{true} \mid a \mid \varphi \land \varphi \mid \neg \varphi \mid \Diamond \varphi \mid \varphi \mathbin{U} \varphi, \quad a \in AP \)

▶ alternative expression of more formulae

\[
\begin{align*}
\varphi_1 \lor \varphi_2 &= \neg(\neg \varphi_1 \land \neg \varphi_2) \\
\varphi_1 \Rightarrow \varphi_2 &= \neg \varphi_1 \lor \varphi_2
\end{align*}
\]

and of two temporal modalities

\[
\begin{align*}
\Diamond \varphi &= \text{true} \mathbin{U} \varphi \\
\Box \varphi &= \neg \Diamond \neg \varphi
\end{align*}
\]
Alternative syntax in the literature

- you may encounter the following notations:

\[
\begin{align*}
X\varphi & : \bigcirc \varphi \\
F\varphi & : \lozenge \varphi \\
G\varphi & : \blacksquare \varphi
\end{align*}
\]

(notation on left-hand side from [CGP99], on right-hand side from [BK08])
Semantics of LTL

\[
TS \models \varphi \iff \forall s \in I : s \models \varphi
\]

where

\[
s \models \varphi \iff \forall \pi \in Paths(s) : \pi \models \varphi
\]
Semantics of LTL

\[ TS \models \varphi \iff \forall s \in I : s \models \varphi \]

where

\[ s \models \varphi \iff \forall \pi \in Paths(s) : \pi \models \varphi \]

and where (cf. LTL syntax)

\[ \pi \models \text{true} \]
\[ \pi \models a \iff a \in L(\pi[0]) \]
\[ \pi \models \varphi \land \psi \iff \pi \models \varphi \land \pi \models \psi \]
\[ \pi \models \neg \varphi \iff \pi \not\models \varphi \]
\[ \pi \models \Diamond \varphi \iff \pi[1..] \models \varphi \]
\[ \pi \models \varphi \cup \psi \iff \exists i \geq 0 : \pi[i..] \models \psi \land \forall 0 \leq j < i : \pi[j..] \models \varphi \]
Alternative semantics of LTL

Let $\varphi$ be an LTL formula over $AP$, inducing the LT property

$$\text{Words}(\varphi) = \{ \sigma \in (2^{AP})^\omega \mid \sigma \models \varphi \}$$

where $(2^{AP})^\omega = 2^{AP} \times 2^{AP} \times \ldots$, and where

$$\sigma \models \text{true}$$

$$\sigma \models a \iff a \in 2^{AP}$$

... 

$TS \models \varphi$ iff $\text{Traces}(TS) \subseteq \text{Words}(\varphi)$
LTL properties for the traffic light model

- how to express the property “the light is infinitely often red” by an LTL formula?
- □◊red
LTL properties for the traffic light model

- how to express the property
  “the light is infinitely often red”
  by an LTL formula?
  ▶ □◊red

- how to express the property
  “once green, the light cannot become red immediately”
  by an LTL formula?
  ▶ □(g\text{reen} \Rightarrow \neg\bigcirc r\text{ed})
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

\[ \pi : 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \ldots \]

- question: \( \pi \models \text{red?} \)
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

  $\pi : 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots$

- question: $\pi \models \text{red}$?

- answer: yes
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

$$\pi : \quad 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots$$

- question: $$\pi \models \circ \circ \text{red}$$?
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

\[ \pi : 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots \]

- question: \( \pi \models igcirc \bigcirc \text{red?} \)

- answer: no
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

\[ \pi: 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots \]

- question: \( \pi \models \text{red} \cup \text{green} \)?
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

$$\pi: 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots$$

- question: $$\pi \models \text{red } \cup \text{ green}$$?

- answer: yes, because $$L(2) = \{\text{red, amber}\}$$
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

\[ \pi : 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots \]

- question: \( \pi \models \Diamond \text{black} \)?
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

$$\pi : \quad \begin{array}{c}
1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \ldots
\end{array}$$

- question: $$\pi \models \Diamond \text{black}$$?

- answer: yes
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

\[ \pi : 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots \]

- question: \( \pi \models \square \neg \text{red}? \)
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

\[ \pi : 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots \]

- question: \( \pi \models \square \neg \text{red} ? \)

- answer: no
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

  $\pi : 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots$

- question: $\pi \models (\Diamond \text{black}) \cup (\Diamond \text{red})$?
Verification of LTL specs is over linear-time paths

- back to the traffic light model, consider the following path:

\[ \pi : 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow \cdots \]

- question: \( \pi \models (♦ \text{black}) \cup (\circ \text{red})? \)

- answer: yes
Classes of LTL specifications

question: what class of LTL formulas capture *invariants*?
Classes of LTL specifications

question: what class of LTL formulas capture *invariants*?

answer: $\Box \varphi$, where $\varphi ::= \text{true} \mid a \mid \varphi \land \varphi \mid \neg \varphi$
Classes of LTL specifications

▶ question: what class of LTL formulas capture *invariants*?

▶ answer: □φ, where φ ::= true | a | φ ∧ φ | ¬φ

▶ example: □¬red
Classes of LTL specifications

- question: how is the class of safety properties characterized?

  answer: “nothing bad ever happens”

  example: “a red light is immediately preceded by amber”

  answer: \( \neg \text{red} \land \Box (\Box \text{red} \Rightarrow \text{amber}) \)
Classes of LTL specifications

- question: how is the class of safety properties characterized?
- answer: “nothing bad ever happens”
Classes of LTL specifications

- question: how is the class of safety properties characterized?
- answer: “nothing bad ever happens”
- example: “a red light is immediately preceded by amber”
- question: how can we express this property in LTL?
Classes of LTL specifications

▶ question: how is the class of safety properties characterized?
▶ answer: “nothing bad ever happens”
▶ example: “a red light is immediately preceded by amber”
▶ question: how can we express this property in LTL?
▶ answer: $\neg \text{red} \land \Box(\Diamond \text{red} \Rightarrow \text{amber})$
Classes of LTL specifications

- question: how is the class of *liveness properties* characterized?
Classes of LTL specifications

- question: how is the class of *liveness properties* characterized?

- answer: “something good eventually happens”
Classes of LTL specifications

▶ question: how is the class of *liveness properties* characterized?

▶ answer: “something good eventually happens”

▶ example: “the light is infinitely often red”

▶ question: how can we express this property in LTL?
Classes of LTL specifications

- question: how is the class of *liveness properties* characterized?

- answer: “something good eventually happens”

- example: “the light is infinitely often red”

- question: how can we express this property in LTL?

- answer: □◊red
Expressiveness of LTL

- are there temporal properties we cannot express in LTL?
- yes, example: “always a state satisfying a can be reached”
Expressiveness of LTL

- are there temporal properties we cannot express in LTL?
- yes, example: “always a state satisfying a can be reached”
- example 2:

\[
\forall \pi \in Paths(TS) : \forall m \geq 0 : \exists \pi' \in Paths(\pi[m]) : \exists n \geq 0 : \pi'[n] \models a.
\]
Expressiveness of LTL

- are there temporal properties we cannot express in LTL?
- yes, example: “always a state satisfying a can be reached”
- example 2:

$$\forall \pi \in \text{Paths}(TS) : \forall m \geq 0 : \exists \pi' \in \text{Paths}(\pi[m]) : \exists n \geq 0 : \pi'[n] \models a.$$
Outline

Course setup

Intro to formal verification

Models - labelled transition systems

Properties as specifications - modal logics

Model checking algorithms
Unfolding of a transition system

- the following transition system

![Diagram of a transition system]

- can be unfolded via its paths as follows...
LTL, a linear temporal logic
CTL, a branching temporal logic
Syntax of CTL

- **state formulae** are defined by

  \[ \Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \varphi \mid \forall \varphi \]

- **path formulae** are defined by

  \[ \varphi ::= \Box \Phi \mid \Phi \lor \Phi \]
Syntax of CTL

- **state formulae** are defined by

  \[ \Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \varphi \mid \forall \varphi \]

- **path formulae** are defined by

  \[ \varphi ::= \Box \Phi \mid \Phi \cup \Phi \]

- two temporal modalities are introduced as

  \[
  \begin{align*}
  \exists \Diamond \Phi &= \exists (\text{true} \cup \Phi) \\
  \forall \Diamond \Phi &= \forall (\text{true} \cup \Phi) \\
  \exists \Box \Phi &= \neg \forall (\text{true} \cup \neg \Phi) \\
  \forall \Box \Phi &= \neg \exists (\text{true} \cup \neg \Phi)
  \end{align*}
  \]
CTL properties for the traffic light model

question: how to express the property
   “each red light is preceded by an amber light”
in CTL?

¬ red ∧ ∀□ (amber ∨ ∀⃝¬ red)
(derive via implication rule)
CTL properties for the traffic light model

▶ question: how to express the property
   “each red light is preceded by an amber light”
in CTL?

▶ answer: $\neg red \land \forall \Box (amber \lor \forall \Diamond \neg red)$
(derive via implication rule)
CTL properties for the traffic light model

question: how to express the property “the light is infinitely often green” in CTL?
CTL properties for the traffic light model

▶ question: how to express the property “the light is infinitely often green” in CTL?

▶ answer: $\forall \square \forall \Diamond \text{green}$
Semantics of CTL

\[ TS \models \Phi \iff \forall s \in I : s \models \Phi \]

where (cf. CTL syntax)

\[ s \models \text{true} \]
\[ s \models a \iff a \in L(s) \]
\[ s \models \Phi \land \Psi \iff s \models \Phi \land s \models \Psi \]
\[ s \models \neg \Phi \iff s \not\models \Phi \]
\[ s \models \exists \varphi \iff \exists \pi \in \text{Paths}(s) : \pi \models \varphi \]
\[ s \models \forall \varphi \iff \forall \pi \in \text{Paths}(s) : \pi \models \varphi \]

and where

\[ \pi \models \Box \Phi \iff \pi[1] \models \Phi \]
\[ \pi \models \Phi \lor \Psi \iff \exists i \geq 0 : \pi[i] \models \Psi \land \forall 0 \leq j < i : \pi[j] \models \Phi \]
Semantics of CTL

\[ TS \models \Phi \iff \forall s \in I : s \models \Phi \]

where (cf. CTL syntax)

- \( s \models true \)
- \( s \models a \iff a \in L(s) \)
- \( s \models \Phi \land \Psi \iff s \models \Phi \land s \models \Psi \)
- \( s \models \neg \Phi \iff s \not\models \Phi \)
- \( s \models \exists \varphi \iff \exists \pi \in \text{Paths}(s) : \pi \models \varphi \)
- \( s \models \forall \varphi \iff \forall \pi \in \text{Paths}(s) : \pi \models \varphi \)

and where

\[ \pi \models \bigcirc \Phi \iff \pi[1] \models \Phi \]
\[ \pi \models \Phi \lor \Psi \iff \exists i \geq 0 : \pi[i] \models \Psi \land \forall 0 \leq j < i : \pi[j] \models \Phi \]

**Satisfaction set** \( \text{Sat}(\Phi) \) is defined by \( \text{Sat}(\Phi) = \{ s \in S \mid s \models \Phi \} \)

Thus,

\[ TS \models \Phi \iff I \subseteq \text{Sat}(\Phi) \]
On the semantics of CTL temporal modalities

- recall that

\[ \exists \Diamond \Phi = \exists (\text{true} \cup \Phi) \]

- question: how is

\[ s \models \exists \Diamond \Phi \]

defined?
On the semantics of CTL temporal modalities

- recall that
  \[ \exists \Diamond \Phi = \exists (\text{true} \lor \Phi) \]

- question: how is \( s \models \exists \Diamond \Phi \) defined?

- answer:
  \[ \exists \pi \in Paths(s) : \exists i \geq 0 : \pi[i] \models \Phi \]
On the semantics of CTL temporal modalities

- recall that
  \[ \forall \Box \Phi = \neg \exists (\text{true} \ U \ \neg \Phi) \]

- question: how is
  \[ s \models \forall \Box \Phi \]
  defined?
On the semantics of CTL temporal modalities

- recall that

\[ \forall \Box \Phi = \neg \exists (true U \neg \Phi) \]

- question: how is

\[ s \models \forall \Box \Phi \]

defined?

- answer:

\[ \forall \pi \in Paths(s) : \forall i \geq 0 : \pi[i] \models \Phi \]
On the semantics of CTL temporal modalities

- recall that
  \[ \forall \Box \Phi = \neg \exists (\text{true} \cup \neg \Phi) \]

- question: how is \( s \models \forall \Box \Phi \) defined?

- answer:
  \[ \forall \pi \in \text{Paths}(s) : \forall i \geq 0 : \pi[i] \models \Phi \]

- (likewise for the other two temporal modalities)
Example of CTL semantics via TS unfolding

question: $\exists\Box\green$?

answer:
Example of CTL semantics via TS unfolding

question: $\exists \Diamond \text{green}$?

answer: yes
Example of CTL semantics via TS unfolding

▶ question: $\forall \Diamond \text{black}$?
▶ answer: no
Example of CTL semantics via TS unfolding

(question: $\forall\lozenge\text{black}$?

(answer: no)
Example of CTL semantics via TS unfolding

question: \( \exists \square \neg \text{black?} \)

answer: yes
Example of CTL semantics via TS unfolding

question: $\exists \square \neg \text{black}$?

answer: yes
Example of CTL semantics via TS unfolding

- question: $\forall \square \text{black}?$
- answer:
Example of CTL semantics via TS unfolding

- question: ∀□black?
- answer: no
Example of CTL semantics via TS unfolding

question: $\exists ((\neg \text{black}) \cup \text{black})$?

answer:
Example of CTL semantics via TS unfolding

- question: $\exists((\neg \text{black}) \cup \text{black})$?
- answer: yes
CTL* is a modal logic encompassing both LTL and CTL
Expressiveness of LTL and CTL

- CTL\* is a modal logic encompassing both LTL and CTL
- other logics are also used (e.g., \(\nu\)-calculus, epistemic l., . . .)
Outline

Course setup

Intro to formal verification

Models - labelled transition systems

Properties as specifications - modal logics

Model checking algorithms
Model checking procedure

- formalised model of a system
- formalised property

does model satisfy property?

- yes
- no, c-example
Model checking procedure over CTL formulae

(labeled) transition system $TS$

CTL formula $\Phi$

$TS \models \Phi$?

yes, $I \subseteq \text{Sat}(\Phi)$

no, $I \not\subseteq \text{Sat}(\Phi)$

c-example
Model checking procedure over LTL formulae

\[(labeled) \text{ transition system } TS \]

\[TS \models \Phi?\]

yes, \[Traces(TS) \subseteq Words(\Phi)\]

no, c-example

LTL formula \(\Phi\)
Model checking CTL: main idea

- recall that $TS \models \Phi$ iff $I \subseteq \text{Sat}(\Phi)$

- basic idea:
  compute $\text{Sat}(\Phi)$ by recursion on the structure of $\Phi$, then compare $\text{Sat}(\Phi)$ with $I$

- alternative view:
  label each state with the subformulae of $\Phi$ that it satisfies (i.e. with the subformulae that are satisfied on that state) until while formula is covered, then check whether the entire $I$ is tagged by $\Phi$
Model checking CTL

- consider CTL formula $\Phi = \exists \bigcirc a \land \exists (b \cup \exists \Box \neg c)$
- structure of $\Phi$ is given by its syntax tree (parse tree)
  - leaves have atomic propositions
  - nodes have operators
Model checking CTL

CTL formulae are defined by

$$\Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \diamond \Phi \mid \exists (\Phi \cup \Phi) \mid \forall \diamond \Phi \mid \forall (\Phi \cup \Phi)$$

Question

What is Sat(a)?
Model checking CTL

CTL *formulae* are defined by

\[ \Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \diamond \Phi \mid \exists (\Phi \lor \Phi) \mid \forall \diamond \Phi \mid \forall (\Phi \lor \Phi) \]

**Question**
What is \( \text{Sat}(a) \)?

**Answer**
\[ \text{Sat}(a) = \{ s \in S \mid a \in L(s) \} \subseteq S \]

**Alternative view**
Label each state \( s \) satisfying \( a \in L(s) \) with “\( a \)”
Model checking CTL

CTL *formulae* are defined by

$$\Phi ::= true \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \bigcirc \Phi \mid \exists (\Phi U \Phi) \mid \forall \bigcirc \Phi \mid \forall (\Phi U \Phi)$$

**Question**

What is $Sat(\Phi \land \Psi)$?
Model checking CTL

CTL *formulae* are defined by

\[ \Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \Diamond \Phi \mid \exists (\Phi U \Phi) \mid \forall \Box \Phi \mid \forall (\Phi U \Phi) \]

**Question**

What is \( Sat(\Phi \land \Psi) \)?

**Answer**

\( Sat(\Phi \land \Psi) = Sat(\Phi) \cap Sat(\Psi) \)

**Alternative view**

Label states, that are labelled with both \( \Phi \) and \( \Psi \), also with \( \Phi \land \Psi \)
Model checking CTL

CTL formulae are defined by

\[ \Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \diamond \Phi \mid \exists (\Phi U \Phi) \mid \forall \diamond \Phi \mid \forall (\Phi U \Phi) \]

Question
What is Sat(\exists \diamond \Phi)?
Model checking CTL

CTL \emph{formulae} are defined by

\[ \Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \circ \Phi \mid \exists(\Phi U \Phi) \mid \forall \circ \Phi \mid \forall(\Phi U \Phi) \]

\textbf{Question}

What is $Sat(\exists \circ \Phi)$?

\textbf{Answer}

$Sat(\exists \circ \Phi) = \{ s \in S \mid Post(s) \cap Sat(\Phi) \neq \emptyset \}$

\textbf{Alternative view}

Labels those states that have a (not necessarily all) direct successor labelled with $\Phi$, with $\exists \circ \Phi$
Model checking CTL

CTL formulae are defined by

\[ \Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \Box \Phi \mid \exists (\Phi \lor \Phi) \mid \forall \Box \Phi \mid \forall (\Phi \lor \Phi) \]

Question
What is \( \text{Sat}(\exists (\Phi \lor \Psi)) \)?
Model checking CTL

CTL formulae are defined by

\[ \Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid \exists \bigcirc \Phi \mid \exists (\Phi \lor \Phi) \mid \forall \bigcirc \Phi \mid \forall (\Phi \lor \Phi) \]

Question
What is \( \text{Sat}(\exists (\Phi \lor \Psi)) \)?

Proposition
\( \text{Sat}(\exists (\Phi \lor \Psi)) \) is the smallest subset \( T \) of \( S \) such that

(a) \( \text{Sat}(\Psi) \subseteq T \), and

(b) if \( s \in \text{Sat}(\Phi) \) and \( \text{Post}(s) \cap T \neq \emptyset \) then \( s \in T \)

Notice that: \( \exists (\Phi \lor \Psi) = \Psi \lor (\Phi \land \exists \bigcirc \exists (\Phi \lor \Psi)) \);
pick \( T = \exists (\Phi \lor \Psi) \), then \( T \) satisfies conditions above

- condition above can be implemented via reachability procedure
Computation of $Sat(\exists (\Phi U \Psi))$

- Condition above can be implemented via reachability procedure
- Existence of least fixpoint via lattice theory (Tarski-Knaster’s fixpoint theorem)
- Over finite TS, procedure completes in finite time
- But valid also for more complicated (infinite-state) models
Model checking CTL – comments

- in its explicit version, boils down to state-space reachability analysis, over the formula tree
- clearly, this wouldn’t scale to models of high cardinality
- in its practice, CTL MC leverages a symbolic implementation
- plus a number of other techniques, such as BMC, SAT-based counterexample generation, induction, interpolations, abstractions/refinements

- nuSMV is a CTL model checker widely used
Outline

Course setup

Intro to formal verification

Models - labelled transition systems

Properties as specifications - modal logics

Model checking algorithms
Model checking LTL: main idea

- LTL model checking algorithm takes
  - a model $TS$ and
  - a formula $\varphi$

and returns

- yes if $TS \models \varphi$
- no and a counter-example if $TS \not\models \varphi$
Model checking LTL: main idea

- LTL model checking algorithm takes
  - a model $TS$ and
  - a formula $\varphi$

  and returns
  - yes if $TS \models \varphi$
  - no and a counter-example if $TS \not\models \varphi$

- here we look into the *automata-based* approach
- (alternatively, *tableau* construction, and indeed, many more alternatives!)
Model checking LTL: essential steps

- consider model $TS$, LTL property $\varphi$
- qualitatively, $TS \models \varphi$ if for all the paths $\pi$ of $TS$ it holds that $\pi \models \varphi$, namely if $\text{Trace}(\pi) \in \text{Words}(\varphi)$
- equivalently, $TS$ admits no path $\pi$ such that $\pi \models \neg \varphi$; if such a path $\pi$ exists, it is a counterexample for $TS$ over $\varphi$
Model checking LTL: essential steps

▶ consider model $TS$, LTL property $\varphi$

▶ qualitatively, $TS \models \varphi$ if for all the paths $\pi$ of $TS$ it holds that $\pi \models \varphi$, namely if $\text{Trace}(\pi) \in \text{Words}(\varphi)$

▶ equivalently, $TS$ admits no path $\pi$ such that $\pi \models \neg \varphi$; if such a path $\pi$ exists, it is a counterexample for $TS$ over $\varphi$

▶ more formally,

$$TS \models \varphi \iff \text{Traces}(TS) \subseteq \text{Words}(\varphi)$$
$$\iff \text{Traces}(TS) \cap \left( (2^{AP})^\omega \setminus \text{Words}(\varphi) \right) = \emptyset$$
$$\iff \text{Traces}(TS) \cap \text{Words}(\neg \varphi) = \emptyset$$
Model checking LTL: essential steps

- consider model TS, LTL property \( \varphi \)

- qualitatively, \( TS \models \varphi \) if for all the paths \( \pi \) of TS it holds that \( \pi \models \varphi \), namely if \( \text{Trace}(\pi) \in \text{Words}(\varphi) \)

- equivalently, TS admits no path \( \pi \) such that \( \pi \models \neg \varphi \); if such a path \( \pi \) exists, it is a counterexample for TS over \( \varphi \)

- more formally,

\[
TS \models \varphi \iff \text{Traces}(TS) \subseteq \text{Words}(\varphi) \\
\iff \text{Traces}(TS) \cap \{(2^AP)^\omega \setminus \text{Words}(\varphi)\} = \emptyset \\
\iff \text{Traces}(TS) \cap \text{Words}(\neg \varphi) = \emptyset
\]

- let's look into the entity \( \text{Words}(\neg \varphi) \)
Expressing formulae via models

- let’s look into the entity $Words(\neg \varphi)$
Expressing formulae via models

- let’s look into the entity $Words(\neg \varphi)$

- first, we relate to an LTL formula ($\neg \varphi$) a model, namely an automaton $A_{\neg \varphi}$
- recall that $Words(\neg \varphi) \subseteq (2^{AP})^\omega$ (is $\omega$-regular)
- we need an non-terminating automaton with an alphabet $\Sigma$ on $2^{AP}$: as an option, use of NBA
- then, we look at the language associated to the automaton, $L_\omega(A_{\neg \varphi})$
Expressing formulae via models

- let’s look into the entity $Words(\neg \varphi)$

- first, we relate to an LTL formula $(\neg \varphi)$ a model, namely an automaton $A_{\neg \varphi}$

- recall that $Words(\neg \varphi) \subseteq (2^{AP})^\omega$ (is $\omega$-regular)

- we need an non-terminating automaton with an alphabet $\Sigma$ on $2^{AP}$: as an option, use of NBA

- then, we look at the language associated to the automaton, $L_\omega(A_{\neg \varphi})$

- finally, express check $Traces(TS) \cap Words(\neg \varphi) = \emptyset$ as $Traces(TS) \cap L_\omega(A_{\neg \varphi}) = \emptyset$
Definition
A generalised non-deterministic Büchi automaton $\mathcal{G}$ is a tuple $(Q, \Sigma, \delta, Q_0, \mathcal{F})$, where

- $Q$ is a finite state space,
- $\Sigma$ is an alphabet,
- $\delta: Q \times \Sigma \to 2^Q$ is a transition relation,
- $Q_0 \subseteq Q$ is a set of initial states,
- $\mathcal{F} \subseteq 2^Q$ are the accepting states.
GNBA - accepting run

- NBA $\mathcal{A}$ is obtained as special case where $\mathcal{F} = F \subseteq Q$, that is a GNBA $\mathcal{G}$ is an NBA $\mathcal{A}$ whenever $\mathcal{F}$ is a singleton
- additional special cases:
  - DBA (deterministic, less expressive)
  - NFA (finite runs – different semantics, equiv. to DFA)
GNBA - accepting run

- NBA $\mathcal{A}$ is obtained as special case where $\mathcal{F} = F \subseteq Q$, that is a GNBA $\mathcal{G}$ is an NBA $\mathcal{A}$ whenever $\mathcal{F}$ is a singleton
- additional special cases:
  - DBA (deterministic, less expressive)
  - NFA (finite runs – different semantics, equiv. to DFA)

- for NBA, an accepting run for $\sigma = A_0A_1A_2\ldots \in \Sigma^\omega$ is a sequence $q_0q_1q_2\ldots$, where $q_0 \in Q_0$, $q_i \in Q$, and $\delta(q_i, A_i) = q_{i+1}$; and $q_i \in F$ for infinitely many indices
- for GNBA, last condition becomes: for all $F \in \mathcal{F}$, $q_i \in F$ for infinitely many indices
(G)NBA - examples

\[ \varphi = \Box \Diamond a \]

\[ \varphi = \Box (a \rightarrow \Diamond b) \]

\[ \varphi = \Box (a \rightarrow \Diamond b) \]

\[ \mathcal{A} : \]

\[ \mathcal{A} : \]

\[ \text{(double circles denote accepting states)} \]
Model checking LTL: essential steps

- consider model $TS$, LTL property $\varphi$

\[
TS \models \varphi \iff \text{Traces}(TS) \subseteq \text{Words}(\varphi) \\
\iff \text{Traces}(TS) \subseteq (2^{AP})^\omega \setminus \text{Words}(\neg \varphi) \\
\iff \text{Traces}(TS) \cap \left((2^{AP})^\omega \setminus \text{Words}(\varphi)\right) = \emptyset \\
\iff \text{Traces}(TS) \cap \text{Words}(\neg \varphi) = \emptyset \\
\iff \text{Traces}(TS) \cap \mathcal{L}_\omega(A_{\neg \varphi}) = \emptyset \\
\iff TS \otimes A_{\neg \varphi} \models \Diamond \Box \neg F
\]

- $TS \otimes A_{\neg \varphi}$ is a “product automaton”
- $F$ is set of accepting states of $A_{\neg \varphi}$
Product Automaton - example

- model of pedestrian traffic light (green, red) \( TS \)
- \( \varphi = \Box \Diamond g \), thus \( \neg \varphi = \Diamond \Box \neg g \); build \( A_{\neg \varphi} \)

\[
TS : \quad r \xrightarrow{} g \\
A_{\neg \varphi} : \quad q_0 \xrightarrow{\neg g} q_1 \xrightarrow{g} q_2
\]
Product Automaton - example

- model of pedestrian traffic light (green, red) TS
- $\varphi = \Box \Diamond g$, thus $\neg \varphi = \Diamond \Box \neg g$; build $A_{\neg \varphi}$

$TS : \quad \quad g \quad \quad r$

$A_{\neg \varphi} : \quad \quad q_0 \quad \quad q_1 \quad \quad q_2$

$TS \otimes A_{\neg \varphi} :$

$\langle r, q_0 \rangle; q_0 \quad \quad \langle r, q_1 \rangle; q_1 \quad \quad \langle r, q_2 \rangle; q_2$

$\langle g, q_0 \rangle; q_0 \quad \quad \langle g, q_1 \rangle; q_1 \quad \quad \langle g, q_2 \rangle; q_2$
Product Automaton - example

- model of pedestrian traffic light (green, red) \( TS \)
- \( \varphi = \Box \Diamond g \), thus \( \neg \varphi = \Diamond \Box \neg g \); build \( A_{\neg \varphi} \)

\[
TS : \quad \begin{array}{c}
\rightarrow \quad r \quad \rightarrow \\
\quad \quad g
\end{array}
\]

\[
A_{\neg \varphi} : \quad \begin{array}{c}
\rightarrow \quad \neg g \\
\quad \quad q_0 \quad \quad \rightarrow \\
\quad \quad (q_1) \quad \quad \rightarrow \\
\quad \quad g \quad \quad q_2
\end{array}
\]

\[
TS \otimes A_{\neg \varphi} : \quad \begin{array}{c}
\rightarrow \quad \langle r, q_0 \rangle ; q_0 \\
\quad \quad \langle g, q_0 \rangle ; q_0 \\
\rightarrow \quad \langle r, q_1 \rangle ; q_1 \\
\quad \quad \langle g, q_1 \rangle ; q_1 \\
\rightarrow \quad \langle r, q_2 \rangle ; q_2 \\
\quad \quad \langle g, q_2 \rangle ; q_2
\end{array}
\]

- \( P_{\text{pers}}(A) = \Diamond \Box \neg q_1 \)
- \( TS \otimes A_{\neg \varphi} \models P_{\text{pers}}(A) \Rightarrow TS \models \varphi \)
Comments on LTL model checking algorithm

- again LTL boils down to reachability analysis
- however in the practice . . .
  - on-the-fly procedures
  - symbolic procedures with BDD,
  - SAT-based BMC,
  - use of counter-examples,
  - Craig interpolation and induction,
  - abstractions and refinements
  - compositional verification

- SPIN is an LTL model checker widely used
Outline

Course setup

Intro to formal verification

Models - labelled transition systems

Properties as specifications - modal logics

Model checking algorithms
Support reading from [BK08] for this lecture

ch 2 LTS models

ch 4 LT properties

ch 5 LTL - logic and model checking algorithms

ch 6 CTL - logic and model checking algorithms

ch 7 equivalences and abstractions