Model Checking
Lecture 1
Outline

1 Specifications: logic vs. automata, linear vs. branching, safety vs. liveness
2 Graph algorithms for model checking
3 Symbolic algorithms for model checking
4 Pushdown systems
Model checking, narrowly interpreted:

Decision procedures for checking if a given Kripke structure is a model for a given formula of a modal logic.
Why is this of interest to us?

Because the dynamics of a discrete system can be captured by a Kripke structure.

Because some dynamic properties of a discrete system can be stated in modal logics.

→

Model checking = System verification
Model checking, generously interpreted:

Algorithms, rather than proof calculi, for system verification which operate on a system model (semantics), rather than a system description (syntax).
There are many different model-checking problems:
for different (classes of) system models
for different (classes of) system properties
A specific model-checking problem is defined by

\[ I \models S \]

“implementation” (system model)

“specification” (system property)

“satisfies”, “implements”, “refines” (satisfaction relation)
A specific model-checking problem is defined by

I ≤ S

more detailed

“implementation”
(system model)

more abstract

“specification”
(system property)

“satisfies”, “implements”, “refines”
(satisfaction relation)
Characteristics of system models which favor model checking over other verification techniques:

- **ongoing input/output behavior**
  (not: single input, single result)

- **concurrency**
  (not: single control flow)

- **control intensive**
  (not: lots of data manipulation)
Examples

- control logic of hardware designs
- communication protocols
- device drivers
Paradigmatic example:

mutual-exclusion protocol

\[ \text{loop} \]
\[ \text{out: } x1 := 1; \text{ last := 1} \]
\[ \text{req: await } x2 = 0 \text{ or } \text{last = 2} \]
\[ \text{in: } x1 := 0 \]
\[ \text{end loop.} \]

\[ \ || \ ]

\[ \text{loop} \]
\[ \text{out: } x2 := 1; \text{ last := 2} \]
\[ \text{req: await } x1 = 0 \text{ or } \text{last = 1} \]
\[ \text{in: } x2 := 0 \]
\[ \text{end loop.} \]
Model-checking problem

$I \models S$

- system model
- system property
- satisfaction relation
Model-checking problem

$I \models S$

- system model
- system property
- satisfaction relation
Important decisions when choosing a system model

- state-based vs. event-based
- interleaving vs. true concurrency
- synchronous vs. asynchronous interaction
- etc.
Particular combinations of choices yield

CSP
Petri nets
I/O automata
Reactive modules
etc.
While the choice of system model is important for ease of modeling in a given situation, the only thing that is important for model checking is that the system model can be translated into some form of state-transition graph.
State-transition graph

- $Q$: set of states
  - $\{q_1, q_2, q_3\}$
- $A$: set of atomic observations
  - $\{a, b\}$
- $\rightarrow \subseteq Q \times Q$: transition relation
  - $q_1 \rightarrow q_2$
- $[\cdot]: Q \rightarrow 2^A$: observation function
  - $[q_1] = \{a\}$

set of observations
Mutual-exclusion protocol

loop
  out:  x1 := 1; last := 1
  req:  await x2 = 0 or last = 2
  in:   x1 := 0
end loop.

||

loop
  out:  x2 := 1; last := 2
  req:  await x1 = 0 or last = 1
  in:   x2 := 0
end loop.

P1

P2
3 \cdot 3 \cdot 2 \cdot 2 \cdot 2 = 72 \text{ states}
The translation from a system description to a state-transition graph usually involves an exponential blow-up !!!

\[ \text{e.g., } n \text{ boolean variables } \Rightarrow 2^n \text{ states} \]

This is called the “state-explosion problem.”
Finite state-transition graphs don’t handle:
- recursion (need pushdown models)
- process creation

State-transition graphs are not necessarily finite-state

We will talk about some of these issues in a later lecture.
Model-checking problem

\[ I \models S \]

- system model
- system property
- satisfaction relation
Three important decisions when choosing system properties:

1. automata vs. logic
2. branching vs. linear time
3. safety vs. liveness
Three important decisions when choosing system properties:

1. automata vs. logic
2. branching vs. linear time
3. safety vs. liveness

The three decisions are orthogonal, and they lead to substantially different model-checking problems.
Three important decisions when choosing system properties:

1. automata vs. logic
2. branching vs. linear time
3. safety vs. liveness

The three decisions are orthogonal, and they lead to substantially different model-checking problems.
Safety vs. liveness

Safety: something “bad” will never happen
Liveness: something “good” will happen
(but we don’t know when)
Safety vs. liveness for sequential programs

Safety: the program will never produce a wrong result ("partial correctness")

Liveness: the program will produce a result ("termination")
Safety vs. liveness for sequential programs

Safety: the program will never produce a wrong result ("partial correctness")

Liveness: the program will produce a result ("termination")
Safety vs. liveness for state-transition graphs

Safety: those properties whose violation always has a finite witness
(if something bad happens on an infinite run, then it happens already on some finite prefix)

Liveness: those properties whose violation never has a finite witness
(no matter what happens along a finite run, something good could still happen later)
Run: \[ q_1 \rightarrow q_3 \rightarrow q_1 \rightarrow q_3 \rightarrow q_1 \rightarrow q_2 \rightarrow q_2 \rightarrow \]

Trace: \[ a \rightarrow b \rightarrow a \rightarrow b \rightarrow a \rightarrow a,b \rightarrow a,b \rightarrow \]
State-transition graph $S = (Q, A, \rightarrow, [])$

Finite runs: $\text{finRuns}(S) \subseteq Q^*$

Infinite runs: $\text{infRuns}(S) \subseteq Q^\omega$

Finite traces: $\text{finTraces}(S) \subseteq (2^A)^*$

Infinite traces: $\text{infTraces}(S) \subseteq (2^A)^\omega$
Safety: the properties that can be checked on finRuns

Liveness: the properties that cannot be checked on finRuns
Safety: the properties that can be checked on finRuns

Liveness: the properties that cannot be checked on finRuns

(they need to be checked on infRuns)

This is much easier.
Example: Mutual exclusion

It cannot happen that both processes are in their critical sections simultaneously.
Example: Mutual exclusion

It cannot happen that both processes are in their critical sections simultaneously.

Safety
Example: Bounded overtaking

Whenever process P1 wants to enter the critical section, then process P2 gets to enter at most once before process P1 gets to enter.
Example: Bounded overtaking

Whenever process P1 wants to enter the critical section, then process P2 gets to enter at most once before process P1 gets to enter.

Safety
Example: Starvation freedom

Whenever process P1 wants to enter the critical section, provided process P2 never stays in the critical section forever, P1 gets to enter eventually.
Example: Starvation freedom

Whenever process P1 wants to enter the critical section, provided process P2 never stays in the critical section forever, P1 gets to enter eventually.

Liveness
infRuns $\Rightarrow$ finRuns
infRuns $\Rightarrow$ finRuns

\[\text{closure}\]

*finite branching*
For state-transition graphs, all properties are safety properties!
Example: Starvation freedom

Whenever process P1 wants to enter the critical section, *provided* process P2 never stays in the critical section forever, P1 gets to enter eventually.

Liveness
Fairness constraint:

the green transition cannot be ignored forever
Without fairness: \[ \text{infRuns} = q_1 (q_3 q_1)^* q_2^\omega \cup (q_1 q_3)^\omega \]

With fairness: \[ \text{infRuns} = q_1 (q_3 q_1)^* q_2^\omega \]
Two important types of fairness

1. Weak (Buchi) fairness:
   a specified set of transitions cannot be enabled forever without being taken

2. Strong (Streett) fairness:
   a specified set of transitions cannot be enabled infinitely often without being taken
Strong fairness
Weak fairness
Fair state-transition graph \( S = ( Q, A, \rightarrow, [], WF, SF) \)

- \( WF \) set of weakly fair actions
- \( SF \) set of strongly fair actions

where each \textbf{action} is a subset of \( \rightarrow \)
Weak fairness comes from modeling concurrency

\[
\text{loop } x := 0 \text{ end loop.} \quad \parallel \quad \text{loop } x := 1 \text{ end loop.}
\]

\[
\begin{tikzpicture}
\node [circle, draw] (x0) at (0,0) {$x=0$};
\node [circle, draw] (x1) at (1,0) {$x=1$};
\draw [->, green] (x0) .. controls (-0.5,-0.5) and (-0.5,0.5) .. (x0);
\draw [->, green] (x0) .. controls (0.5,-0.5) and (0.5,0.5) .. (x0);
\draw [->, red] (x1) .. controls (-0.5,-0.5) and (-0.5,0.5) .. (x1);
\draw [->, red] (x1) .. controls (0.5,-0.5) and (0.5,0.5) .. (x1);
\end{tikzpicture}
\]

Weakly fair action
Weakly fair action
Strong fairness comes from modeling choice

\[
\text{loop } m:
\begin{align*}
    n: & \quad x:=0 \mid x:=1 \\
\end{align*}
\text{end loop.}
\]
Weak fairness is sufficient for asynchronous models ("no process waits forever if it can move").

Strong fairness is necessary for modeling resource contention.

**Strong fairness makes model checking more difficult.**
Fairness changes only infRuns, not finRuns.

\[ \downarrow \]

Fairness can be ignored for checking safety properties.
Two remarks

The vast majority of properties to be verified are safety.

While nobody will ever observe the violation of a true liveness property, fairness is a useful abstraction that turns complicated safety into simple liveness.
Three important decisions when choosing system properties:

1. automata vs. logic
2. branching vs. linear time
3. safety vs. liveness

The three decisions are orthogonal, and they lead to substantially different model-checking problems.
Fair state-transition graph \( S = ( Q, A, \rightarrow, [], WF, SF ) \)

Finite runs: \( \text{finRuns}(S) \subseteq Q^* \)

Infinite runs: \( \text{infRuns}(S) \subseteq Q^\omega \)

Finite traces: \( \text{finTraces}(S) \subseteq (2^A)^* \)

Infinite traces: \( \text{infTraces}(S) \subseteq (2^A)^\omega \)
Branching vs. linear time

Linear time: the properties that can be checked on infTraces

Branching time: the properties that cannot be checked on infTraces
Same traces \{axb, axc\}
Different runs \{q_0 q_1 q_3, q_0 q_2 q_4\}, \{q_0 q_1 q_3, q_0 q_1 q_4\}
Linear-time:
In all traces, an $x$ must happen immediately followed by $b$
Linear-time:
In all traces, an x must happen immediately followed by b or c
Branching-time:
An $x$ must happen immediately following which $a$ $b$ may happen and $a$ $c$ may happen
Same traces, different runs (different trace trees)
Three important decisions when choosing system properties:

1. automata vs. logic
2. branching vs. linear time
3. safety vs. liveness

The three decisions are orthogonal, and they lead to substantially different model-checking problems.
Logics

Safety

Liveness

Linear

LTL

Branching

STL

CTL
STL (Safe Temporal Logic)

- safety (only finite runs)
- branching
Defining a logic

1. Syntax:
   What are the formulas?

2. Semantics:
   What are the models?
   Does model $M$ satisfy formula $\varphi$? $M \models \varphi$
Propositional logics:

1. boolean variables \((a,b)\) & boolean operators \((\land,\neg)\)
2. model = truth-value assignment for variables

Propositional modal (e.g., temporal) logics:

1. ... & modal operators \((\Box,\Diamond)\)
2. model = set of (e.g., temporally) related prop. models
Propositional logics:

1. boolean variables \((a,b)\) & boolean operators \((\wedge,\neg)\)
2. model = truth-value assignment for variables

Propositional modal \((e.g.,\text{ temporal})\) logics:

1. ... & modal operators \((\Box,\Diamond)\)
2. model = set of \((e.g.,\text{ temporally})\) related prop. models

atomic observations

state-transition graph (“Kripke structure”)
STL Syntax

\[ \phi ::= a \mid \phi \land \phi \mid \neg \phi \mid \exists \Diamond \phi \mid \phi \exists U \phi \]

boolean variable (atomic observation)

boolean operators

modal operators
STL Model

\((K, q)\)

- state-transition graph (Kripke structure)
- state of \(K\)
STL Semantics

\[(K,q) \models a \quad \text{iff} \quad a \in [q]\]

\[(K,q) \models \varphi \land \psi \quad \text{iff} \quad (K,q) \models \varphi \text{ and } (K,q) \models \psi\]

\[(K,q) \models \neg \varphi \quad \text{iff} \quad \text{not } (K,q) \models \varphi\]

\[(K,q) \models \exists \diamond \varphi \quad \text{iff} \quad \text{exists } q' \text{ s.t. } q \rightarrow q' \text{ and } (K,q') \models \varphi\]

\[(K,q) \models \varphi \exists U \psi \quad \text{iff} \quad \text{exists } q = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n.\]

1. for all \(0 \leq i < n\), \((K,q_i) \models \varphi\)
2. \((K,q_n) \models \psi\)
Defined modalities

\( \exists O \)  \hspace{2cm} \text{EX} \hspace{2cm} \text{exists next}

\( \forall O \varphi = \neg \exists O \neg \varphi \)  \hspace{2cm} \text{AX} \hspace{2cm} \text{forall next}

\( \exists U \)  \hspace{2cm} \text{EU} \hspace{2cm} \text{exists until}

\( \exists \Diamond \varphi = \text{true} \exists U \varphi \)  \hspace{2cm} \text{EF} \hspace{2cm} \text{exists eventually}

\( \forall \square \varphi = \neg \exists \Diamond \neg \varphi \)  \hspace{2cm} \text{AG} \hspace{2cm} \text{forall always}

\( \varphi \forall W \psi = \neg ( \neg \psi ) \exists U ( \neg \varphi \land \neg \psi ) \)  \hspace{2cm} \text{AW} \hspace{2cm} \text{forall waiting-for (forall weak-until)}
Exercise

1. Derive the semantics of $\varphi \forall W \psi$:

   $(K,q) \models \varphi \forall W \psi$ iff for all $q_0, q_1, q_2, \ldots$ s.t. $q = q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots$, either for all $i \geq 0$, $(K,q_i) \models \varphi$, or exists $n \geq 0$ s.t.
   
   1. for all $0 \leq i < n$, $(K,q_i) \models \varphi$
   2. $(K,q_n) \models \psi$

2. Derive the semantics of $\neg ( (\neg \psi) \exists U (\neg \varphi))$:

   $(K,q) \models \neg ( (\neg \psi) \exists U (\neg \varphi))$ iff ???
(K,q) |= \varphi \bigwedge W \psi

For all executions starting from q, \psi is satisfied at or before a (the first) violation of \varphi.

(K,q) |= \varphi \bigwedge W \psi
\iff
(K,q) |= \neg ( (\neg \psi) \exists U (\neg \varphi \land \neg \psi))
\iff
\neg (\text{exists } q = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n.
\text{ for all } 0 \leq i < n. (K,q_i) |= \neg \psi \text{ and } (K,q_n) |= \neg \varphi \land \neg \psi)
\iff
\text{for all } q = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n.
\text{exists } 0 \leq i < n. (K,q_i) |= \psi \text{ or } (K,q_n) |= \varphi \lor \psi
\iff
\text{for all } q = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n.
\text{exists } 0 \leq i \leq n. (K,q_i) |= \psi \text{ or } (K,q_n) |= \varphi
\iff
\text{for all } q = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n.
(K,q_n) |= \neg \varphi \Rightarrow \text{exists } 0 \leq i \leq n. (K,q_i) |= \psi
Important safety properties

Invariance \( \forall \Box \ a \)

Sequencing \( a \ W b \ W c \ W d \)
\( = a \ W (b \ W (c \ W d)) \)
Important safety properties: mutex protocol

Invariance: $\forall \Box \neg (pc1=\text{in} \land pc2=\text{in})$

Sequencing: $\forall \Box (pc1=\text{req} \Rightarrow (pc2 \neq \text{in}) \land \forall W (pc2=\text{in}) \land \forall W (pc2 \neq \text{in}) \land \forall W (pc1=\text{in}))$
Branching properties

Deadlock freedom: $\forall \Box \exists \Diamond \text{true}$

Possibility: $\forall \Box (a \Rightarrow \exists \Diamond b)$

$\forall \Box (pc1=\text{req} \Rightarrow \exists \Diamond (pc1=\text{in}))$
CTL (Computation Tree Logic)

- safety & liveness
- branching time

[Clarke & Emerson; Queille & Sifakis 1981]
CTL Syntax

\( \phi ::= a \mid \phi \land \phi \mid \neg \phi \mid \exists \phi \mid \phi \exists U \phi \mid \exists \square \phi \)
CTL Model

\(( K, q )\)

**fair state-transition graph**

**state of** **K**
CTL Semantics

\((K,q) \models \exists \square \varphi \iff \text{exist } q_0, q_1, \ldots \text{ s.t.}
\)

1. \(q = q_0 \rightarrow q_1 \rightarrow \ldots\) is an infinite fair run
2. for all \(i \geq 0\), \((K,q_i) \models \varphi\)
Defined modalities

\[ \exists \Box \quad \text{EG} \quad \text{exists always} \]

\[ \forall \Diamond \, \varphi = \neg \exists \Box \neg \varphi \quad \text{AF} \quad \text{forall eventually} \]

\[ \varphi \exists W \psi = (\varphi \exists U \psi) \lor (\exists \Box \varphi) \]

\[ \varphi \forall U \psi = (\varphi \forall W \psi) \land (\forall \Diamond \psi) \]
Important liveness property

Response

∀□ (a ⇒ ∀◊ b)

∀□ (pc1=req ⇒ ∀◊ (pc1=in))