Software Verification
Exercise class 5:
Model Checking

Carlo A. Furia
Recap of definitions and results
Finite State Automata: Syntax

Def. Nondeterministic Finite State Automaton (FSA):

a tuple \([\Sigma, S, I, \rho, F]\):

- \(\Sigma\): finite nonempty (input) alphabet
- \(S\): finite nonempty set of states
- \(I \subseteq S\): set of initial states
- \(F \subseteq S\): set of accepting states
- \(\rho: S \times \Sigma \rightarrow 2^S\): transition function
Def. An accepting run of an FSA $A = [\Sigma, S, I, \rho, F]$ over input word $w = w(1) w(2) \ldots w(n) \in \Sigma^*$ is a sequence $r = r(0) r(1) r(2) \ldots r(n) \in S^*$ of states such that:

- it starts from an initial state: $r(0) \in I$
- it ends in an accepting state: $r(n) \in F$
- it respects the transition function: $r(i+1) \in \rho(r(i), w(i))$ for all $0 \leq i < n$
Finite State Automata: Semantics

Def. Any FSA $A = [\Sigma, S, I, \rho, F]$ defines

a set of input words $\langle A \rangle$:

$$\langle A \rangle \triangleq \{ w \in \Sigma^* \mid \text{there is an accepting run of } A \text{ over } w \}$$

$\langle A \rangle$ is called the language of $A$
Def. Propositional Linear Temporal Logic (LTL) formulae are defined by the grammar:

\[
F ::= p \mid \neg F \mid F \land G \mid X F \mid F U G
\]

with \( p \in P \) any atomic proposition from a fixed set \( P \).

Temporal (modal) operators:
- next: \( X F \)
- until: \( F U G \)
- release: \( F R G \equiv \neg (\neg F U \neg G) \)
- eventually: \( \Diamond F \equiv \text{True} U F \)
- always: \( \Box F \equiv \neg \Diamond \neg F \)

Propositional connectives:
- not: \( \neg F \)
- and: \( F \land G \)
- or: \( F \lor G \equiv \neg (\neg F \land \neg G) \)
- implies: \( F \Rightarrow G \equiv \neg F \lor G \)
- iff: \( F \Leftrightarrow G \equiv (F \Rightarrow G) \land (G \Rightarrow F) \)
Def. A word \( w = w(1) \ldots w(n) \in (2^P)^* \) satisfies an LTL formula \( F \) at position \( 1 \leq i \leq n \), denoted \( w, i \models F \), under the following conditions:

- \( w, i \models p \) iff \( p \in w(i) \)
- \( w, i \models \neg F \) iff \( w, i \not\models F \) does not hold
- \( w, i \models F \land G \) iff both \( w, i \models F \) and \( w, i \models G \) hold
- \( w, i \models X F \) iff \( i < n \) and \( w, i+1 \models F \)
  - i.e., \( F \) holds in the next step
- \( w, i \models F U G \) iff for some \( i \leq j \leq n \) it is: \( w, j \models G \) and for all \( i \leq k < j \) it is \( w, k \models F \)
  - i.e., \( F \) holds until \( G \) will hold
For derived operators:

- \( w, i \models \Diamond F \) iff for some \( i \leq j \leq n \) it is: \( w, j \models F \)
  - i.e., \( F \) holds \textit{eventually} (in the future)

- \( w, i \models \Box F \) iff for all \( i \leq j \leq n \) it is: \( w, j \models F \)
  - i.e., \( F \) holds \textit{always} (in the future)
Def. Satisfaction:

\[ w \models F \triangleq w, 1 \models F \]

i.e., word \( w \) satisfies formula \( F \) initially

Def. Any LTL formula \( F \) defines a set of words \( \langle F \rangle \):

\[ \langle F \rangle \triangleq \{ w \in (2^P)^* \mid w \models F \} \]

\( \langle F \rangle \) is called the language of \( F \)
Automata-theoretic Model Checking

An semantic view of the Model Checking problem:

- **Given**: a finite-state automaton $A$ and a temporal-logic formula $F$
- if $\langle A \rangle \cap \langle \neg F \rangle$ is empty then any run of $A$ satisfies $F$
- if $\langle A \rangle \cap \langle \neg F \rangle$ is not empty then some run of $A$ does not satisfy $F$
  - any member of the nonempty intersection $\langle A \rangle \cap \langle \neg F \rangle$ is a counterexample

How to check $\langle A \rangle \cap \langle \neg F \rangle = \emptyset$ algorithmically (given $A$, $F$)?

Combination of three different algorithms:

- **LTL2FSA**: given LTL formula $F$ build automaton $a(F)$ such that $\langle F \rangle = \langle a(F) \rangle$
- **FSA-Intersection**: given automata $A$, $B$ build automaton $C$ such that $\langle A \rangle \cap \langle B \rangle = \langle C \rangle$
- **FSA-Emptiness**: given automaton $A$ check whether $\langle A \rangle = \emptyset$ is the case
Exercises:

Semantics of derived operators
Prove that the satisfaction relation

\[ w, i \models \bigcirc F \]

for eventually, defined as:

\[ \bigcirc F \triangleq \text{True} \cup F \]

is equivalent to:

for some \( i \leq j \leq n \) it is: \( w, j \models F \)
LTL derived operators: eventually

\[ w, i \models \Diamond F \]

iff

\[ w, i \models \text{True} \mathbf{U} F \] \quad \text{(definition of eventually)}

iff

for some \( i \leq j \leq n \) it is: \( w, j \models F \)

and for all \( i \leq k < j \) it is \( w, k \models \text{True} \)

(definition of until)

iff

for some \( i \leq j \leq n \) it is: \( w, j \models F \)

(simplification of \( A \) and True)
LTL derived operators: always

Prove that the satisfaction relation

$$w, i \models \Box F$$

for always, defined as:

$$\Box F \equiv \neg \Diamond \neg F$$

is equivalent to:

for all $$i \leq j \leq n$$ it is: $$w, j \models F$$
LTL derived operators: always

\( w, i \models \Box F \)

iff

\( w, i \models \neg \Diamond \neg F \) (definition of always)

iff

\( w, i \models \Diamond \neg F \) is not the case (definition of not)

iff

it is not the case that: for some \( i \leq j \leq n \) it is: \( w, j \models \neg F \) (semantics of eventually)

iff

for all \( i \leq j \leq n \) it is not the case that \( w, j \models \neg F \) (semantics of quantifiers: pushing negation inward)

iff

for all \( i \leq j \leq n \): it is not the case that it is not the case that \( w, j \models F \) (semantics of negation)

iff

for all \( i \leq j \leq n \) it is: \( w, j \models F \) (simplification of double negation)
Exercises:

Evaluate LTL formulas on automata
Does the property hold?

□ (start ⇒ ◊ stop)
Does the property hold?

Yes:

- Whenever `start` occurs we reach state `closed-cooking`.
- We must eventually exit state `closed-cooking` to reach the only accepting state `closed-off`.
- State `closed-cooking` can be exited only if `stop` occurs.
Does the property hold?

□ ◊ turn_off

Diagram with states and transitions:
- Start state: closed cooking
  - Transition: start → closed cooking
  - Transition: stop → closed
- State: closed
  - Transition: pull → open
  - Transition: push → closed
- State: open
  - Transition: pull → open
  - Transition: push → closed
- Transition: turn_off → closed

Turn off state is marked.
Does the property hold?

No:

counterexample:

pull push
Does the property hold?

□ ◊ (turn_off ∨ push)
Does the property hold?

\[ \square \Diamond (\text{turn\_off} \lor \text{push}) \]

Yes:

- every accepting run eventually goes back to state closed-off
- state closed-off can be reached only if either turn\_off or push occurs
- the empty word is also compliant with the semantics of the always operator
Does the property hold?

◊ (turn_off ∨ push)
Does the property hold?

◊ (turn_off ∨ push)

No:

● counterexample: the empty word

(compare the semantics of existential quantification against universal quantification)
Does the property hold?

□ False

◊ (turn_off ∨ push)
Does the property hold?

[Diagram showing a state transition diagram with states: closed off, open off, closed on, open on, closed cooking, pull, push, turn_off, turn_on, stop, start, cook.]

- □ False

\( \lor \) (turn\_off \lor push)

Yes:

- “always False” means that False holds at every step in the word: it is satisfied precisely by the empty word.
- If the word is not empty, then it must end with turn\_off or push, thus it satisfies the other disjunct.
Does the property hold?

\[
\text{turn\_on } \bigcup \text{ start } \lor \text{ pull } \bigcup \text{ push}
\]
Does the property hold?

\[ \text{turn\_on } U \text{ start } \lor \text{ pull } U \text{ push} \]

No:
- counterexample: the empty word
- counterexample: \text{turn\_on } \text{ turn\_off}
- counterexample: \text{turn\_on } \text{ pull } \text{ push } \text{ turn\_off}
Does the property hold?

□ ( start ⇒
(cook U ◊turn_off) )

---

Diagram:

- **closed off**
- **open off**
- **closed on**
- **open on**
- **closed cooking**

Actions:
- pull
- push
- turn_off
- turn_on
- stop
- start
- cook
Does the property hold?

\[ \square ( \text{start} \Rightarrow (\text{cook} \cup \Diamond \text{turn}_{-} \text{off}) ) \]

Yes:

- Once \text{start} occurs, \text{turn}_{-} \text{off} must occur eventually.
- Hence “eventually \text{turn}_{-} \text{off}” is the case right after \text{start} occurs.
- \text{cook} can occur right after \text{start} occurs, zero or more times.
Exercises:

Equivalence of LTL formulas
Equivalence of formulas

Prove that $\lozenge$ is idempotent, that is:

$\lozenge \lozenge q$

is equivalent to:

$\lozenge q$
Equivalence of formulas

\( w, i \models \diamond \diamond q \)

iff

for some \( i \leq j \leq n \) it is: \( w, j \models \diamond q \)

(semantics of eventually)

iff

for some \( i \leq j \leq n \) it is: for some \( j \leq h \leq n \) it is: \( w, h \models q \)

(semantics of eventually)

iff

for some \( i \leq j \leq h \leq n \) it is: \( w, h \models q \)

(merging of intervals)

iff

for some \( i \leq h \leq n \) it is: \( w, h \models q \)

(dropping \( j \), a fortiori)

iff

\( w, i \models \diamond q \)

(semantics of eventually)
Equivalence of formulas

Prove that:

\( p \cup \lozenge q \)

is equivalent to:

\( \lozenge q \)
Equivalence of formulas: $\Rightarrow$ direction

\[ w, i \models p \mathbf{U} \Diamond q \]
iff

for some $i \leq j \leq n$ it is: $w, j \not\models \Diamond q$
and for all $i \leq k < j$ it is $w, k \not\models p$

(semantics of until)

implies

for some $i \leq j \leq n$ it is: $w, j \not\models \Diamond q$ (a fortiori)
iff

for some $i \leq j \leq n$ it is: for some $j \leq h \leq n$ it is: $w, h \not\models q$

(seminaics of eventually)
iff

for some $i \leq h \leq n$ it is: $w, h \not\models q$

(simplification of range of quantification)
iff

\[ w, i \not\models \Diamond q \]  

(seminaics of eventually)
Equivalence of formulas: $\iff$ direction

$w, i \models \diamond q$

iff

for some $i \leq j \leq i$: $w, j \not\models \diamond q$  
(singleton range of quantification)

iff

for some $i \leq j \leq i$: $w, j \not\models \diamond q$  
and True  
(semantics of and)

iff

for some $i \leq j \leq i$: $w, j \not\models \diamond q$

and for all $i \leq k < j= i$ it is $w, k \models p$  
(semantics of universally quantified empty range)

implies

for some $i \leq j \leq n$: $w, j \not\models \diamond q$

and for all $i \leq k < j$ it is $w, k \models p$  
(a fortiori)

iff

$w, i \models p \cup \diamond q$  
(semantics of until)
Exercises:
Automata-theoretic model-checking
(on paper)
Automata-based model checking

Let us prove by model checking that it's not a property of the automaton

\[ \text{\textbf{turn\_off}} \]
Build an automaton with the same language as:

\(!!(\Box \Diamond \text{turn\_off})!

Let us start from the unnegated formula:

\(\Box \Diamond \text{turn\_off}\)

and then complement the states of the automaton.
LTL2FSA

\[ \square \Diamond \text{turn\_off} \]

\[ \neg \text{turn\_off} \text{ turn\_off} \]

\[ \neg \text{turn\_off} \]

\[ \text{turn\_off} \]
\neg( ◻ ◊ {\text{turn\_off}} )
FSA Intersection

[Diagram]

- Nodes: closed off, open off, closed on, open on, closed cooking, start, stop
- Edges: pull, push, turn_on, turn_off

- States: A, B
- Transitions: ¬turn_off, turn_off
FSA-Emptiness: node reachability

Any accepting run on the intersection automaton is a counterexample to the LTL formula being a property of the automaton.

- pull push
- pull push pull push
- ...

![Automaton Diagram](image-url)