Software Verification
Exercise class:
Model Checking
Exercises:
Semantics of derived operators
Prove that the satisfaction relation

\[ w, i \vDash <> F \]

for **eventually**, defined as:

\[ <> F \triangleq True \cup F \]

is equivalent to:

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

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

iff

\[ w, i \models \text{True} \mathbin{U} F \]

(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

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

for always, defined as:

\[ [] F \triangleq \neg \leftrightarrow \neg F \]

is equivalent to:

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

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

iff

\[ w, i \models \neg <-> \neg F \]  
* (definition of always)

iff

\[ w, i \models <-> \neg F \] 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?

\[ [\text{start} \implies \Leftrightarrow \text{stop}] \]
Does the property hold?

\[
\begin{array}{c}
\text{closed off} \\
\text{turn off} \\
\text{turn on} \\
\text{closed on} \\
\text{stop} \\
\text{start} \\
\text{closed cooking} \\
\text{closed-cooking} \\
\text{open off} \\
\text{push} \\
\text{pull} \\
\text{push} \\
\text{pull} \\
\text{open on} \\
\text{on} \\
\text{closed-cooking} \\
\text{closed off} \\
\text{off} \\
\end{array}
\]

\[
\text{[\text{start} \Rightarrow \leftrightarrow \text{stop}]}
\]

Yes:
- whenever \text{start} occurs we reach state \text{closed-cooking}
- we must eventually exit state \text{closed-cooking} to reach the only accepting state \text{closed-off}
- state \text{closed-cooking} can be exited only if \text{stop} occurs
Does the property hold?

\[
\begin{array}{c}
\text{closed} \\
\text{off} \\
\text{push} \\
\text{turn_off} \\
\text{turn_on} \\
\text{closed} \\
\text{on} \\
\text{stop} \\
\text{start} \\
\text{closed} \\
\text{cooking} \\
\text{pull} \\
\text{push} \\
\text{off} \\
\text{on} \\
\text{cook}
\end{array}
\]

\[] \leftrightarrow \text{turn_off}
Does the property hold?

No:
- counterexample: pull push
Does the property hold?

\[ \square \leftrightarrow (\text{turn\_off} \lor \text{push}) \]
Does the property hold?

\[ 
\left[ \right] \leftrightarrow \left( \text{turn\_off} \lor \text{push} \right) 
\]

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?

\[ \langle\rangle \text{(turn\_off \lor \text{push})} \]
Does the property hold?

\[
\langle (\text{turn\_off} \lor \text{push}) \rangle
\]

No:
- counterexample: the empty word

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

\[
[] \text{False} \\
\lor \\
\leftrightarrow (\text{turn\_off} \lor \text{push})
\]
Does the property hold?

[ ] 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?

![Diagram]

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

No:

- counterexample: the **empty** word
- counterexample: `turn_on turn_off`
- counterexample: `turn_on pull push turn_off`

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

\[
[] ( \text{start} \Rightarrow (
\text{cook} \cup \leftrightarrow \text{turn\_off})
) \]

Diagram:

- States: closed, open, closed cooking, on, off
- Transitions: pull, push, turn_on, turn_off, start, stop

Legend:
- Closed: off, closed, cooking
- Open: off, on
- Transition Actions: pull, push
- Start condition: [] (start => (cook \cup \leftrightarrow turn\_off))
Does the property hold?

Yes:
- once start occurs, turn_off must occur eventually
- hence “eventually turn_off” is the case right after start occurs
- cook can occur right after start occurs, one or more times

\[ (\text{start} \Rightarrow (\text{cook} \cup \text{turn\_off})) \]
Exercises:
Equivalence of LTL formulas
Equivalence of formulas

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

$\leftrightarrow \leftrightarrow q$

is equivalent to:

$\leftrightarrow q$
Equivalence of formulas

\( w, i \models \langle\rangle q \) iff

\( \text{for some } i \leq j \leq n \text{ it is: } w, j \models \langle\rangle q \) (semantics of eventually)

iff

\( \text{for some } i \leq j \leq n \text{ it is: for some } j \leq h \leq n \text{ it is: } w, h \models q \) (semantics of eventually)

iff

\( \text{for some } i \leq j \leq h \leq n \text{ it is: } w, h \models q \) (merging of intervals)

iff

\( \text{for some } i \leq h \leq n \text{ it is: } w, h \models q \) (dropping j, a fortiori)

iff

\( w, i \models \langle\rangle q \) (semantics of eventually)
Equivalence of formulas

Prove that:

\( p \leftrightarrow q \)

is equivalent to:

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

\[ w, i \models p \mathcal{U} q \]

iff

\[ \text{for some } i \leq j \leq n \text{ it is: } w, j \not\models q \]

and for all \( i \leq k < j \) it is \( w, k \not\models p \)

implies

\[ \text{for some } i \leq j \leq n \text{ it is: } w, j \not\models q \]  

(a fortiori)

iff

\[ \text{for some } i \leq j \leq n \text{ it is: for some } j \leq h \leq n \text{ it is: } w, h \not\models q \]  

(semantics of eventually)

iff

\[ \text{for some } i \leq h \leq n \text{ it is: } w, h \not\models q \]  

(simplification of range of quantification)

iff

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

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

$w, i \models <> q$

iff

for some $i \leq j \leq i$: $w, j \models <> q$  (singleton range of quantification)

iff

for some $i \leq j \leq i$: $w, j \models <> q$ and True  (semantics of and)

iff

for some $i \leq j \leq i$: $w, j \models <> q$

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

implies

for some $i \leq j \leq n$: $w, j \models <> q$

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

iff

$w, i \models p U <> q$  (semantics of until)
Exercises:
Automata-theoretic model-checking (on paper)
Let us prove by model checking that it's not a property of the automaton
Build an automaton with the same language as:

\neg ( [] <> \text{turn\_off} )

Let us start from the unnegated formula:

[] <> \text{turn\_off}

and then complement the states of the automaton
\neg ( [] \leftrightarrow \text{turn\_off} )
FSA Intersection

Diagram showing the states and transitions for two FSA systems labeled A and B. The transitions include:
- From closed off to open off: pull
- From closed off to turn_off
- From turn_on to closed on
- From closed on to stop
- From stop to closed cooking: cook
- From closed cooking to start
- From start to closed off: push
- From open off to push
- From turn_off to turn_off
- From ¬turn_off to turn_off

The diagram also shows the intersection of these two systems, marked with an 'X'.
FSA Intersection
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
- ...

![Diagram showing a finite state automaton with states and transitions labeled with 'closed off', 'open off', 'closed on', 'open on', 'closed cooking', and actions 'pull', 'push', 'turn on', 'turn off', 'start', 'stop', and 'cook'.]