02246 - Model Checking

$$M \models \Phi ?$$

Lecture 02 - Linear-Time Temporal Logic

Technical University of Denmark

System

refine/implement

Requirements

abstract model

$M \models \Phi$?

some logic here

some smart algorithm here

Finite-state Transition System

yes ☺

may be 😐

no ☹

Key points of this lecture

Temporal Logics as an extension of propositional logics to reason about transition systems.

Linear-time Temporal Logic (LTL) as a logic to reason about traces.

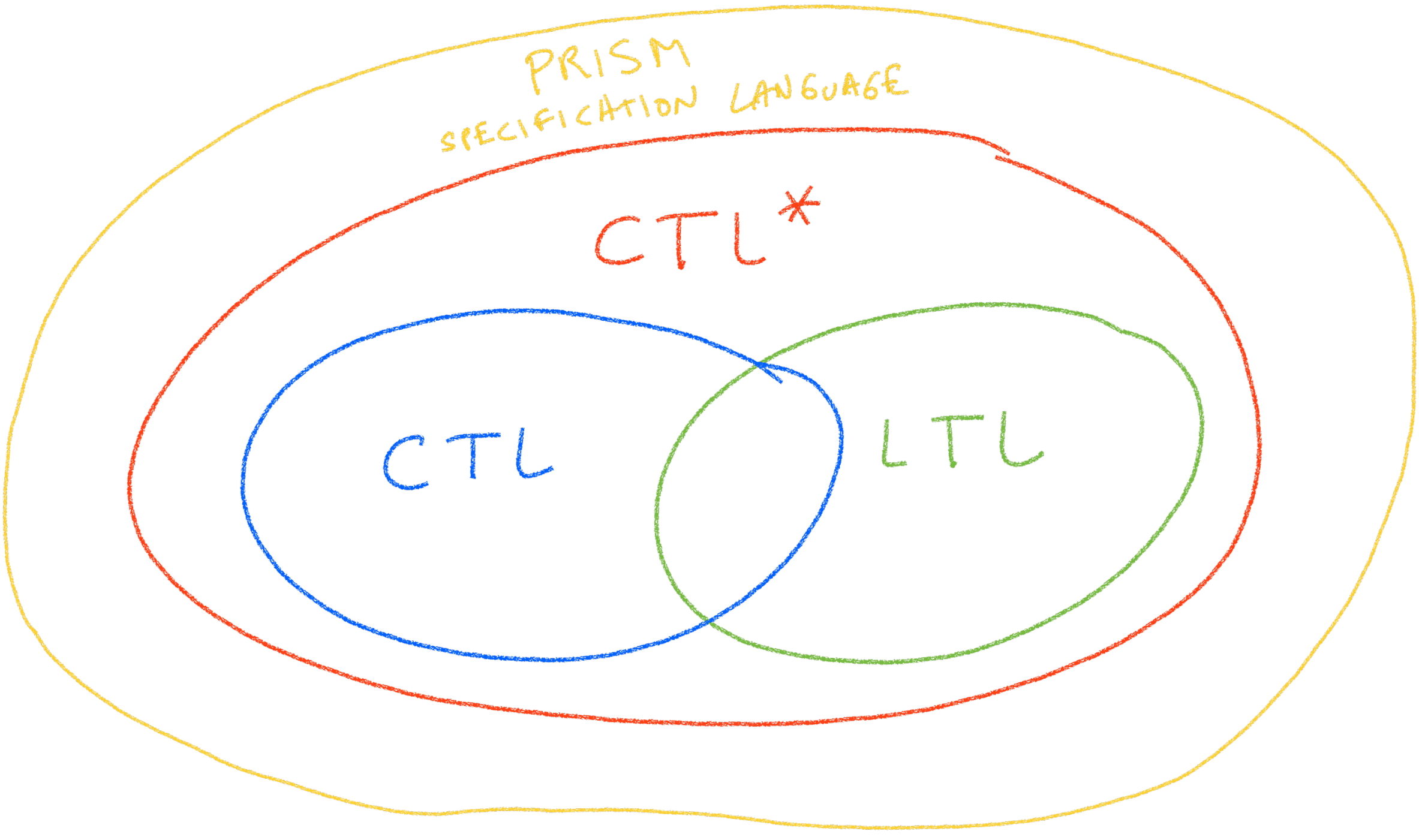
Grammars for LTL.

Formal semantics of LTL: satisfaction relation for paths.

Derived operators and equivalences between LTL formulas.

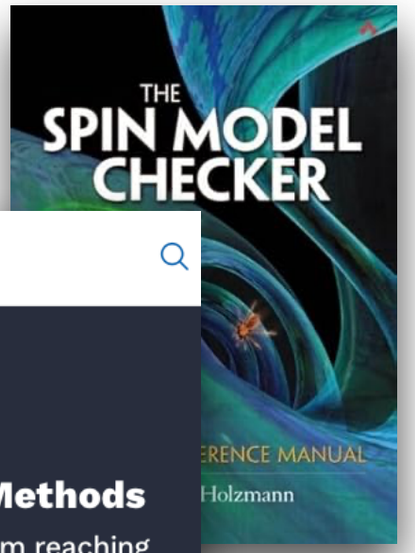
How to write LTL formulas in PRISM.

Families of logics



Why study LTL?

Model checkers (PRISM, SPIN, TLA+, etc.)



COMMUNICATIONS OF THE ACM

RESEARCH AND ADVANCES

Architecture and Hardware

Contributed articles

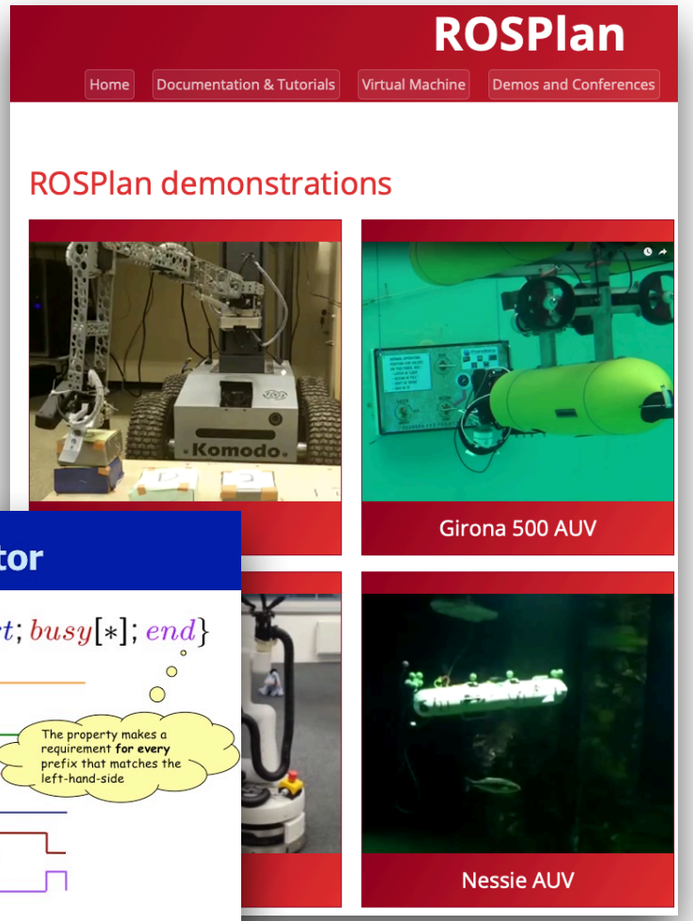
How Amazon Web Services Uses Formal Methods

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

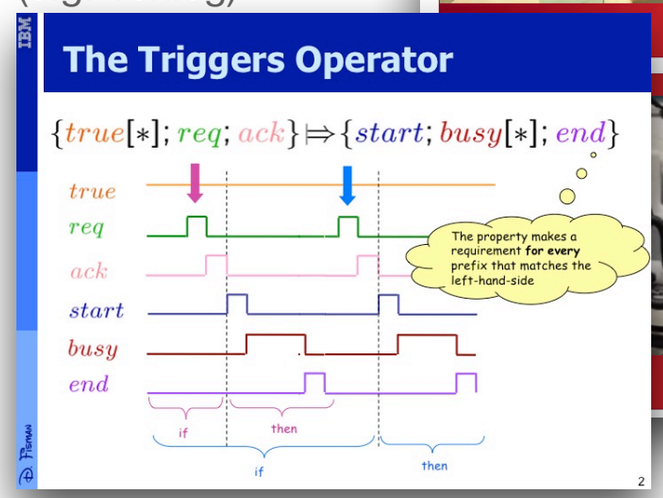
By Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff

Posted Apr 1 2015

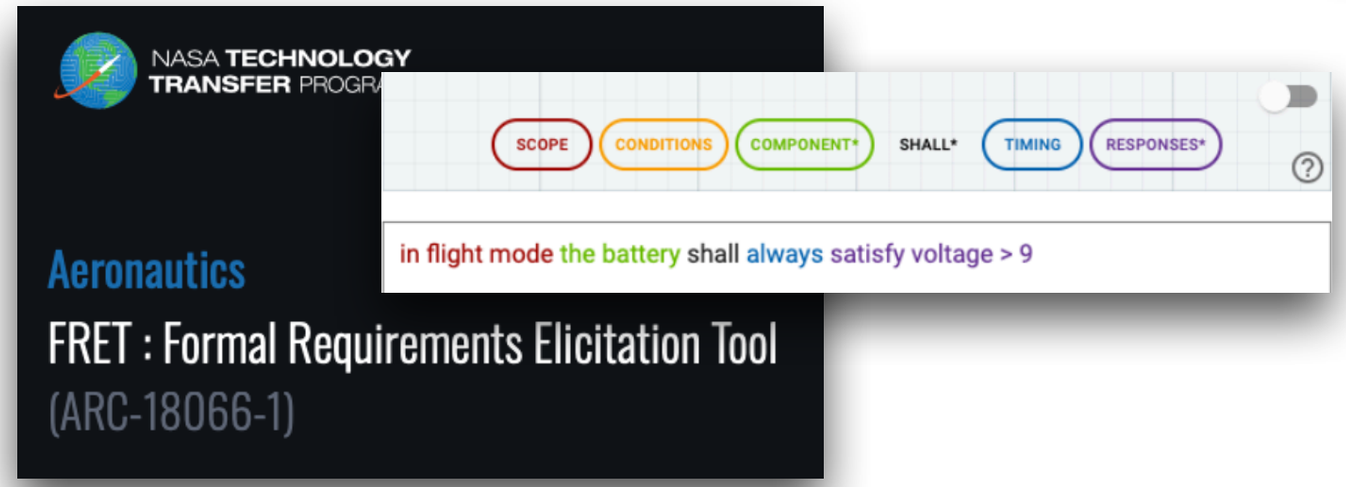
AI planners for autonomous systems (e.g. PDDL, ROS)



HDL verification (e.g. Verilog)



Requirements in critical-safety domains (e.g. NASA's FRETS)



Declarative process models (see courses & research @DTU on process mining, process-driven development, etc.) -> ask Andrey :)

existence	1..* a	◇a	a is executed at least once
absence	0 a	¬◇a	a cannot be executed at all
absence 2	0..1 a	¬◇(a ∧ ○◇a)	a can be executed at most once
choice;	a ◇ b	◇(a ∨ b)	At least one between a and b must be executed
responded existence	a ● b	◇a → ◇b;	If a is executed, then b has also to be executed (either before or later);
response	a ●→ b	□(a → ○◇b)	Whenever a is executed, then b must be executed later

Reading material

Sections 5 (intro), 5.1.1-5.1.4 of Chapter 5 of “Principles of Model Checking”

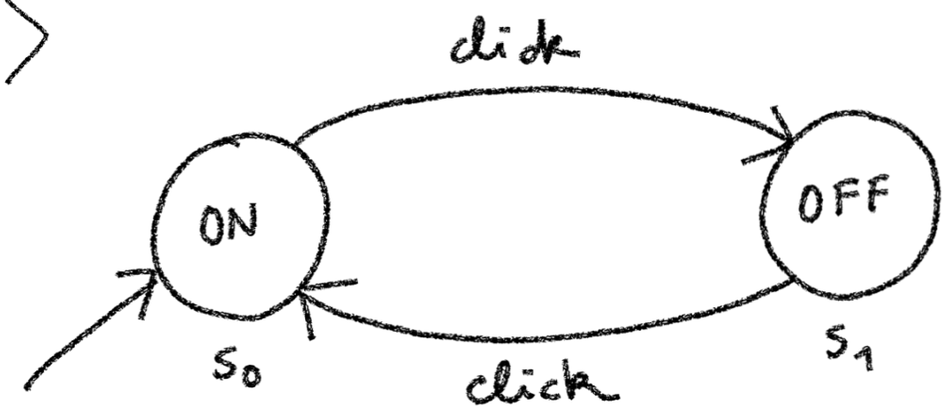
Transition Systems

Def. A transition system is a tuple

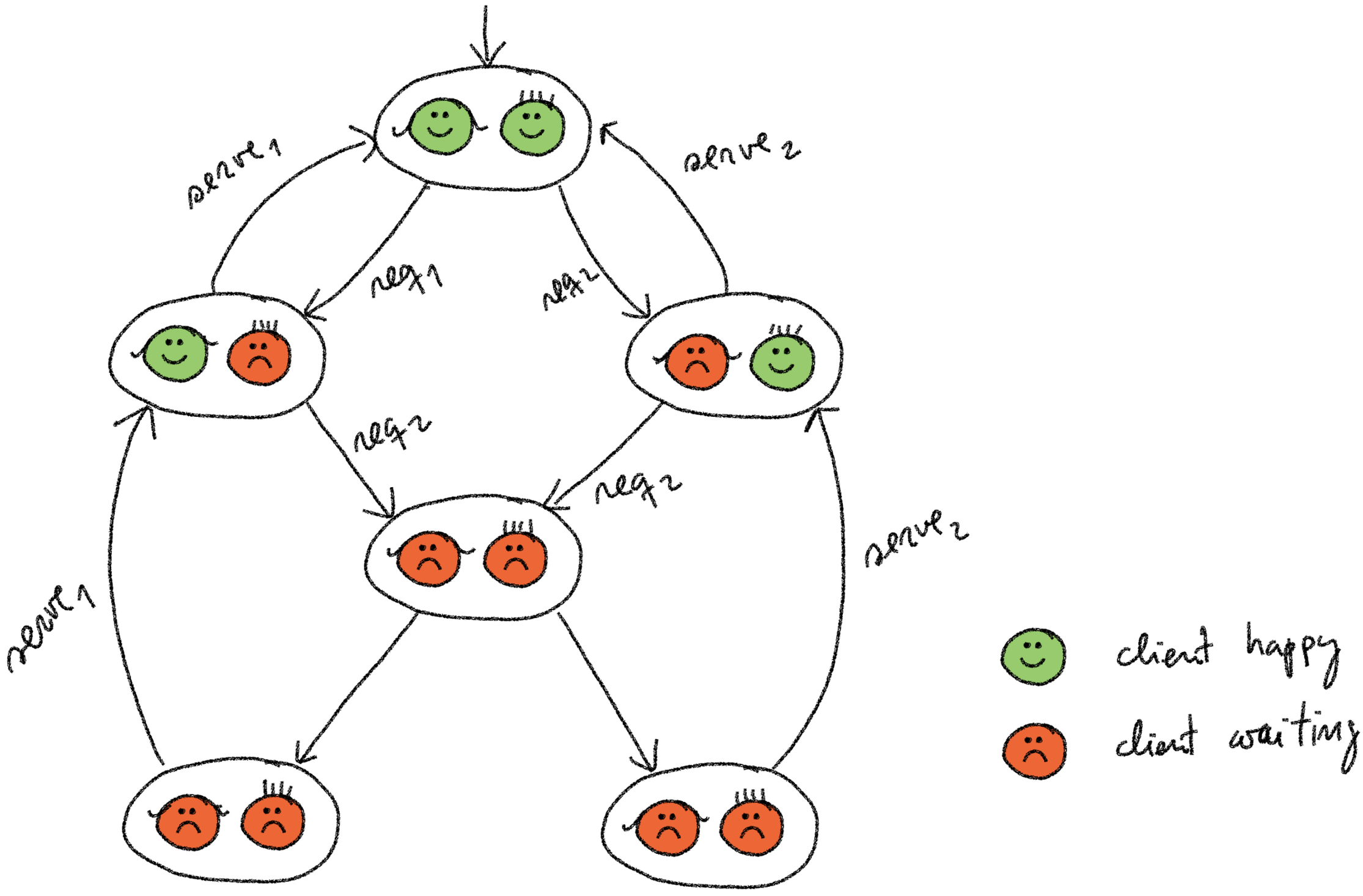
$$\langle S, A, \rightarrow, L, AP, I \rangle$$

such that

- S is a set of states
- A is a set of "Actions"
- $\rightarrow \subseteq S \times A \times S$ is a set of transitions
- $L : S \longrightarrow 2^{AP}$ is a labelling function
- AP is a finite state of "Atomic Propositions"
- $I \subseteq S$ is the set of initial states

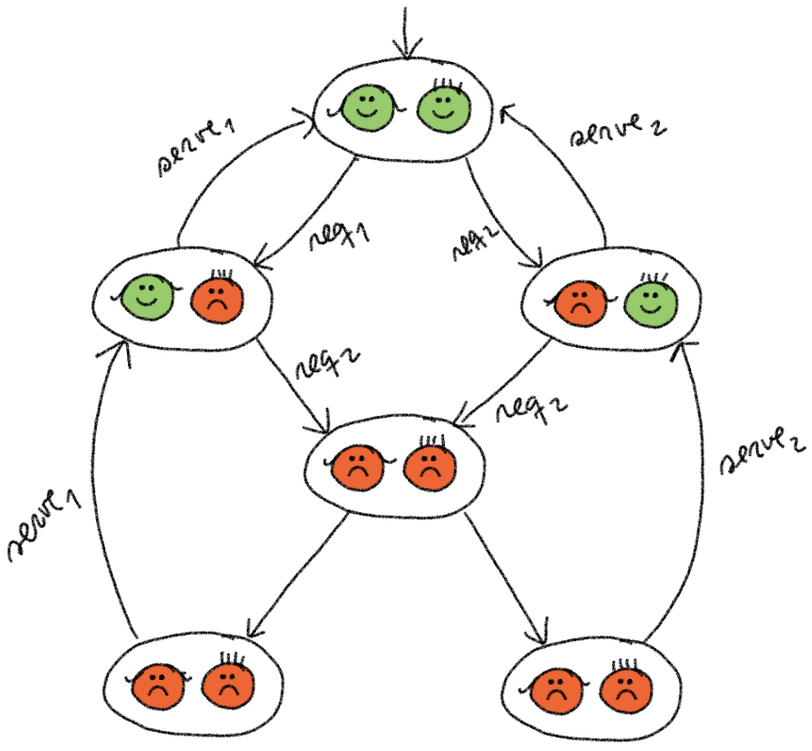


Example: 2 clients, 1 server

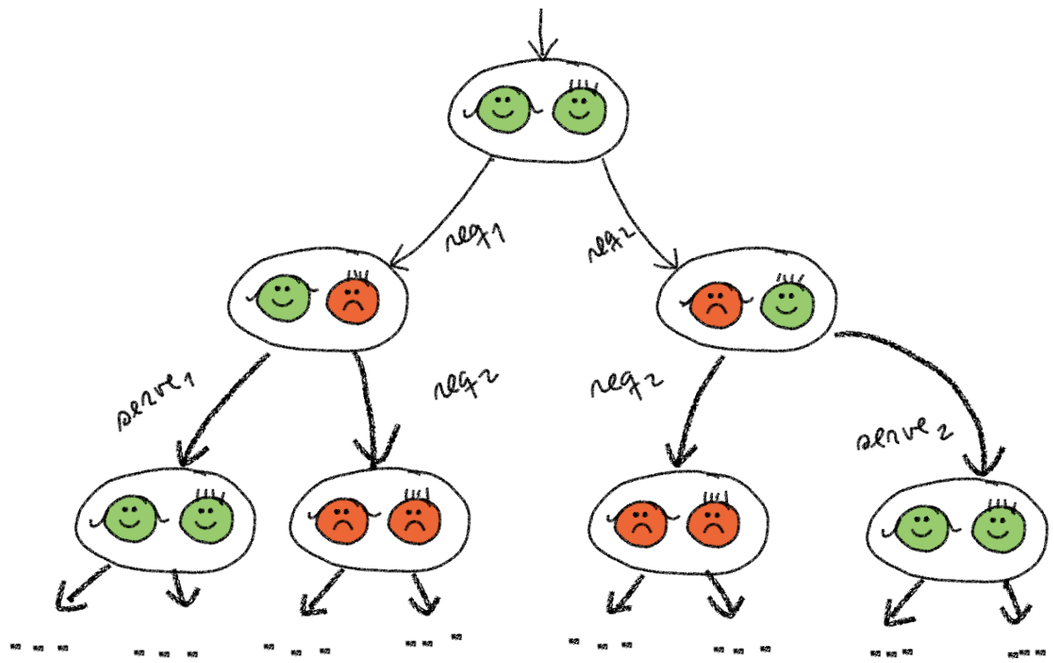


NOTE: model uploaded as `simple-scheduler.prism` on Learn. You can use PRISM to check the example we will see in class

Traces & Computation Trees

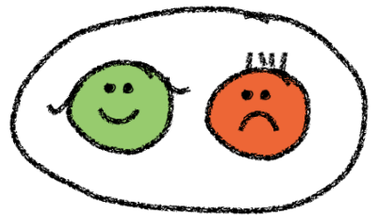


computation tree



Example of trace

Propositional Logic

We can use propositional logic to reason about a state 

$$true$$

$$false = \neg true$$

$$p$$

$$\neg \phi$$

$$\phi_1 \vee \phi_2$$

$$\phi_1 \wedge \phi_2$$

$$\phi_1 \rightarrow \phi_2$$

$$\ldots$$

the state satisfies p

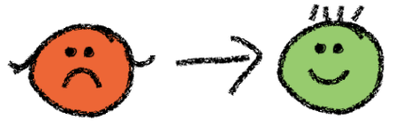
the state does not satisfy ϕ

the state satisfies at least one of ϕ_1, ϕ_2

the state satisfies both of ϕ_1, ϕ_2

if the state satisfies ϕ_1
then it also satisfies ϕ_2

Some examples



Derived operators and grammar for propositional logic

We don't need all operators, some can be derived

$$true$$

$$false = \neg true$$

$$p$$

$$\neg \phi$$

$$\phi_1 \vee \phi_2$$

$$\phi_1 \wedge \phi_2 \equiv \neg(\neg \phi_1 \vee \neg \phi_2)$$

$$\phi_1 \rightarrow \phi_2 \equiv \neg \phi_1 \vee \phi_2$$

We can then use a minimalistic grammar for formulas

$$\phi ::= true \mid p \mid \neg \phi \mid \phi_1 \vee \phi_2$$

BONUS: Recap on propositional logic…

Propositional logic: introduction

Main components

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$
- ¬ negation (“not”) $\neg p$

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$
- $\neg$ negation (“not”) $\neg p$
- $\vee$ disjunction (“or”) $p \vee q$

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$
- $\neg$ negation (“not”) $\neg p$
- $\vee$ disjunction (“or”) $p \vee q$
- $\wedge$ conjunction (“and”) $p \wedge q$

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$
- $\neg$ negation (“not”) $\neg p$
- $\vee$ disjunction (“or”) $p \vee q$
- $\wedge$ conjunction (“and”) $p \wedge q$
- $\rightarrow$ implication (“if..then..”) $p \rightarrow q$

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$
- $\neg$ negation (“not”) $\neg p$
- $\vee$ disjunction (“or”) $p \vee q$
- $\wedge$ conjunction (“and”) $p \wedge q$
- $\rightarrow$ implication (“if..then..”) $p \rightarrow q$
- $\leftrightarrow$ bi-implication (“if and only if”) $p \leftrightarrow q$

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$
- $\neg$ negation ("not") $\neg p$
- $\vee$ disjunction ("or") $p \vee q$
- $\wedge$ conjunction ("and") $p \wedge q$
- $\rightarrow$ implication ("if..then..") $p \rightarrow q$
- $\leftrightarrow$ bi-implication ("if and only if") $p \leftrightarrow q$

Operators

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$
- ¬ negation (“not”) $\neg p$
- ∨ disjunction (“or”) $p \vee q$
- ∧ conjunction (“and”) $p \wedge q$
- → implication (“if..then..”) $p \rightarrow q$
- ↔ bi-implication (“if and only if”) $p \leftrightarrow q$

Operators

A propositional formula is a composition of propositional variables and operators

Propositional logic: introduction

Main components

- propositional (or Boolean) variables $p, q, r, p_1, p_2, \ldots$
- truth constants $\bot$, $\top$
- ¬ negation (“not”) $\neg p$
- ∨ disjunction (“or”) $p \vee q$
- ∧ conjunction (“and”) $p \wedge q$
- → implication (“if..then..”) $p \rightarrow q$
- ↔ bi-implication (“if and only if”) $p \leftrightarrow q$

Operators

A propositional formula is a composition of propositional variables and operators

The **BNF grammar for building propositional formulas**:

$$\varphi ::= p \mid \bot \mid \top \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi)$$

Propositional logic: introduction

Some conventions

- No outer parentheses
- Order of operations: $\neg \succ \wedge \succ \vee \succ \rightarrow , \leftrightarrow$
- $\wedge , \vee , \rightarrow$ are right-associative: $p \rightarrow q \rightarrow r$ denotes $p \rightarrow (q \rightarrow r)$

Propositional logic: semantics

Semantics

- **valuation** (or truth assignment) is a function $v : \{p, q, r, \ldots\} \to \{\mathtt{F}, \mathtt{T}\}$ from propositions to truth values
- It is extended to formulas as follows

$$v(\,\bot\,) = \mathtt{F} \qquad v(\,\top\,) = \mathtt{T} \qquad v(\neg\psi) = \begin{cases} \mathtt{F} & \text{if } v(\varphi) = \mathtt{T} \\ \mathtt{T} & \text{if } v(\varphi) = \mathtt{F} \end{cases}$$

Propositional logic: semantics

Semantics

- **valuation** (or truth assignment) is a function $v : \{p, q, r, \ldots\} \to \{\mathtt{F}, \mathtt{T}\}$ from propositions to truth values
- It is extended to formulas as follows

$$v(\bot) = \mathtt{F} \qquad v(\top) = \mathtt{T} \qquad v(\neg\psi) = \begin{cases} \mathtt{F} & \text{if } v(\varphi) = \mathtt{T} \\ \mathtt{T} & \text{if } v(\varphi) = \mathtt{F} \end{cases}$$

$$v(\psi \vee \varphi) = \begin{cases} \mathtt{F} & \text{if } v(\psi) = v(\varphi) = \mathtt{F} \\ \mathtt{T} & \text{otherwise} \end{cases} \qquad v(\psi \wedge \varphi) = \begin{cases} \mathtt{T} & \text{if } v(\psi) = v(\varphi) = \mathtt{T} \\ \mathtt{F} & \text{otherwise} \end{cases}$$

Propositional logic: semantics

Semantics

- **valuation** (or truth assignment) is a function $v : \{p, q, r, \ldots\} \to \{\mathtt{F}, \mathtt{T}\}$ from propositions to truth values
- It is extended to formulas as follows

$$v(\bot) = \mathtt{F} \qquad v(\top) = \mathtt{T} \qquad v(\neg\psi) = \begin{cases} \mathtt{F} & \text{if } v(\varphi) = \mathtt{T} \\ \mathtt{T} & \text{if } v(\varphi) = \mathtt{F} \end{cases}$$

$$v(\psi \vee \varphi) = \begin{cases} \mathtt{F} & \text{if } v(\psi) = v(\varphi) = \mathtt{F} \\ \mathtt{T} & \text{otherwise} \end{cases} \qquad v(\psi \wedge \varphi) = \begin{cases} \mathtt{T} & \text{if } v(\psi) = v(\varphi) = \mathtt{T} \\ \mathtt{F} & \text{otherwise} \end{cases}$$

$$v(\psi \to \varphi) = \begin{cases} \mathtt{F} & \text{if } v(\psi) = \mathtt{T}, v(\varphi) = \mathtt{F} \\ \mathtt{T} & \text{otherwise} \end{cases} \qquad v(\psi \leftrightarrow \varphi) = \begin{cases} \mathtt{T} & \text{if } v(\psi) = v(\varphi) \\ \mathtt{F} & \text{otherwise} \end{cases}$$

Propositional logic: semantics

Semantics

- **valuation** (or truth assignment) is a function $v : \{p, q, r, \ldots\} \to \{\mathtt{F}, \mathtt{T}\}$ from propositions to truth values
- It is extended to formulas as follows

$$v(\bot) = \mathtt{F} \qquad v(\top) = \mathtt{T} \qquad v(\neg\psi) = \begin{cases} \mathtt{F} & \text{if } v(\varphi) = \mathtt{T} \\ \mathtt{T} & \text{if } v(\varphi) = \mathtt{F} \end{cases}$$

$$v(\psi \vee \varphi) = \begin{cases} \mathtt{F} & \text{if } v(\psi) = v(\varphi) = \mathtt{F} \\ \mathtt{T} & \text{otherwise} \end{cases} \qquad v(\psi \wedge \varphi) = \begin{cases} \mathtt{T} & \text{if } v(\psi) = v(\varphi) = \mathtt{T} \\ \mathtt{F} & \text{otherwise} \end{cases}$$

$$v(\psi \to \varphi) = \begin{cases} \mathtt{F} & \text{if } v(\psi) = \mathtt{T}, v(\varphi) = \mathtt{F} \\ \mathtt{T} & \text{otherwise} \end{cases} \qquad v(\psi \leftrightarrow \varphi) = \begin{cases} \mathtt{T} & \text{if } v(\psi) = v(\varphi) \\ \mathtt{F} & \text{otherwise} \end{cases}$$

Alternative: use satisfaction relation $\models$ (e.g., $v \models \neg\varphi$ iff $v \not\models \varphi$)

Valuations in transition systems

The labelling function defines the valuation function for every state s:

$$v_s(p) = T \quad \text{iff} \quad p \in L(s)$$

Valuations of formulas in a state will be defined with a satisfaction relation $\models$

Truth tables

There are other (less formal, more intuitive) ways to present the semantics

p	$\neg p$
T	
F	

p	q	$p \wedge q$
T	T	
T	F	
F	T	
F	F	

p	q	$p \vee q$
T	T	
T	F	
F	T	
F	F	

p	q	$p \rightarrow q$
T	T	
T	F	
F	T	
F	F	

p	q	$p \leftrightarrow q$
T	T	
T	F	
F	T	
F	F	

Truth tables

There are other (less formal, more intuitive) ways to present the semantics

p	$\neg p$
T	F
F	T

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Computing with truth tables

Truth tables can be used to compute *all possible truth assignments* for a given formula

Exercise: find all valuations for $p \vee (q \wedge \neg z)$

p	q	z	$p \vee (q \wedge \neg z)$
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	

Computing with truth tables

Truth tables can be used to compute *all possible truth assignments* for a given formula

Exercise: find all valuations for $p \vee (q \wedge \neg z)$

p	q	z	$p \vee (q \wedge \neg z)$
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	F
F	T	F	T
F	F	T	F
F	F	F	F

Propositional logic: satisfiability

Formula φ is **satisfiable** if there *exists* valuation v such that $v(\varphi) = \mathrm{T}$

- v is a **model** of φ iff v satisfies φ —> φ is SAT iff φ has a model
- **model()** function in Z3
- unsatisfiable = no models

Formula φ is **valid** if *for all* valuations v it holds that $v(\varphi) = \mathrm{T}$

- alternatively: every v is a **model** of φ (or, $\neg\varphi$ is unsatisfiable)
- not valid = at least one v is not a model

Formulas φ and ψ are **equivalent** ($\varphi \equiv \psi$) if $v(\varphi) = v(\psi)$ for all valuations v

Propositional logic: satisfiability

Exercise: connect the dots

1. **Not valid**
2. **Valid**
3. **SAT**
4. **UNSAT**

A. $(x \to y) \to y$

B. $x \wedge \neg y \wedge (x \to y)$

C. $x \wedge (x \to y) \to y$

D. $x \wedge (x \to y) \leftrightarrow y$

Question: how are validity and SAT related?

DEMO?
Z3? BLACK?

SAT using Z3/Python

colab.research.google.com

02246-2025 playground - Colab

sat-validity-in-Z3 - 02246 Model Che

02246-2025 playground

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[2] ✓ 10s

```
!pip install z3-solver
```

```
Collecting z3-solver
  Downloading z3_solver-4.15.3.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (602 bytes)
Downloading z3_solver-4.15.3.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (29.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 29.1/29.1 MB 64.3 MB/s eta 0:00:00
Installing collected packages: z3-solver
Successfully installed z3-solver-4.15.3.0
```

[7] ✓ 0s

```
from z3 import *

x  = Bool('x')
y  = Bool('y')

#formula = Implies(Implies(x,y),y)
#formula = Implies(And(x,Implies(x,y)),y)
formula = Implies(And(x,Implies(x,y)),y)

# check if formula is sat
print("Checking SAT...")
solver = Solver()
solver.add(formula)
result = solver.check()
if result == sat:
  print("The formula is SATisfiable. Here is a model:")
  m = solver.model()
  print(m)
elif result == unsat:
  print("The formula is UNSATisfiable")
else:
  print("I don't know")
```

SAT using Black

```
alberto@macbookpro ~ % black solve --model --formula "(x -> y) -> y"
SAT
Model:
-t = 0: {y}
```

Let’s start with LTL….

Adding "temporal" operators

Propositional logic is not enough if we want to talk about what is happening along a trace

next	$\bigcirc\phi$	from the next state, the execution satisfies ϕ
eventually or finally	$\Diamond\phi$	some suffix of the execution satisfies ϕ
always	$\Box\phi$	all states in the execution satisfy ϕ
until	$\phi_1 \mathsf{U} \phi_2$	ϕ_2 holds in some state of the execution and until then all states satisfy ϕ_1
	...	

LTL grammar

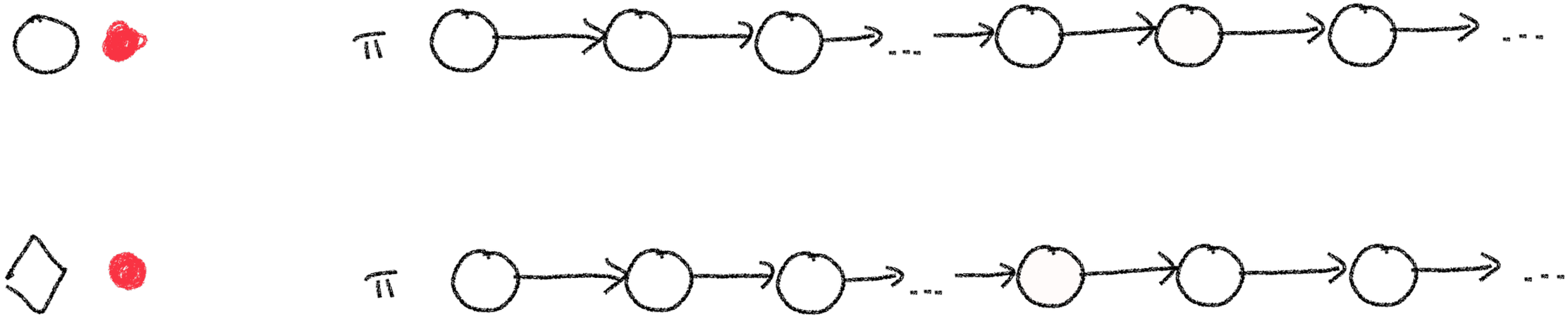
The grammar LTL extends the one of propositional logic temporal operators

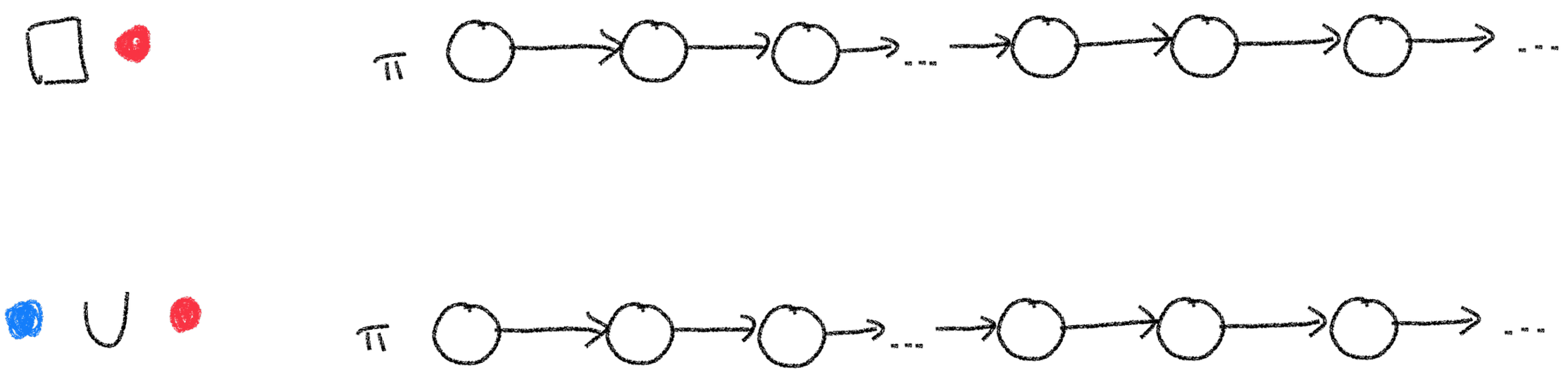
$$\phi ::= true \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc\phi \mid \Diamond\phi \mid \Box\phi \mid \phi_1 \mathsf{U} \phi_2$$

Alternative / equivalent notation

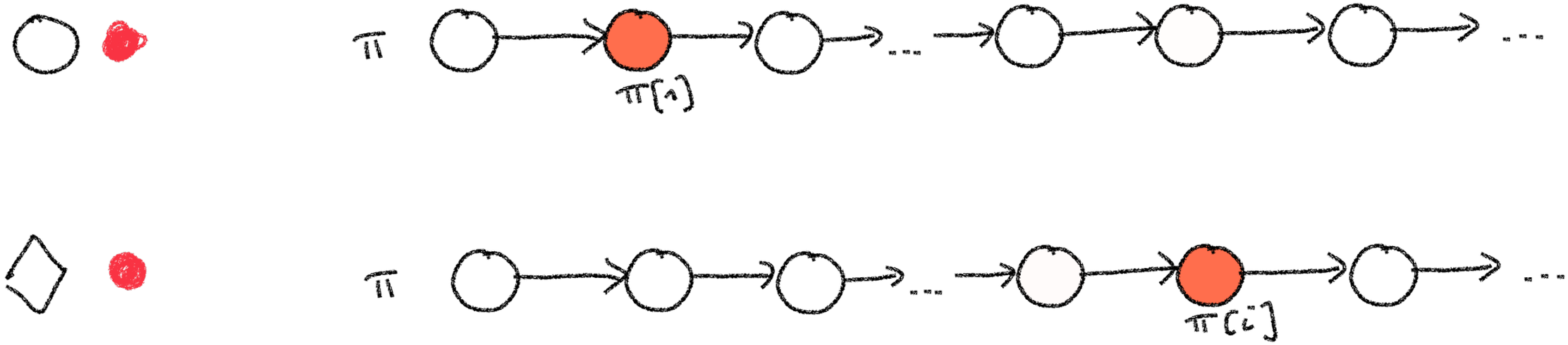
math/LaTeX style		ascii style
$\bigcirc$	X	
$\Diamond$	F	
$\Box$	G	

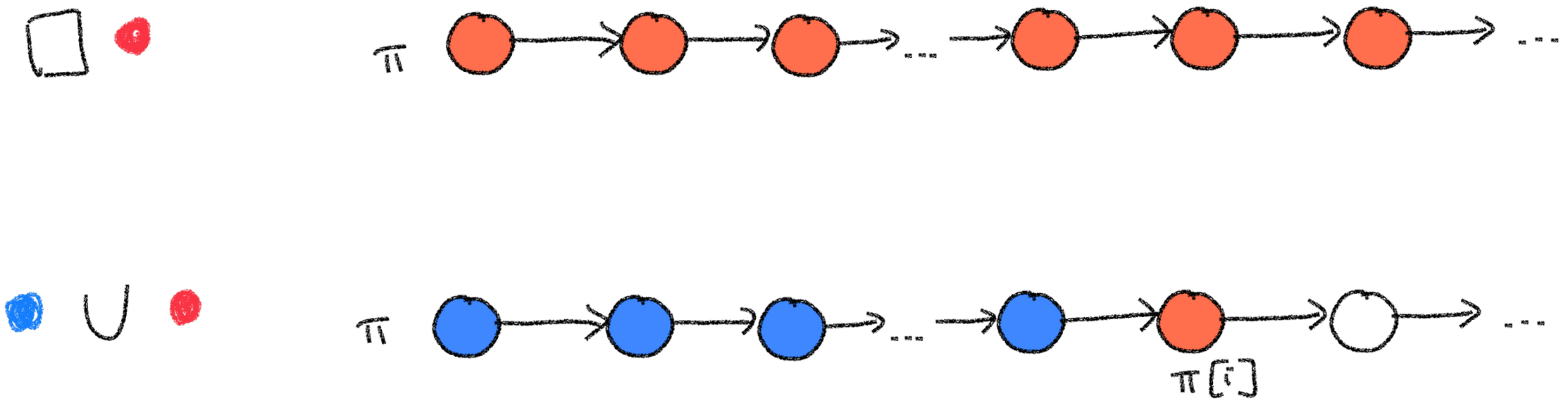
Intuition of temporal operators



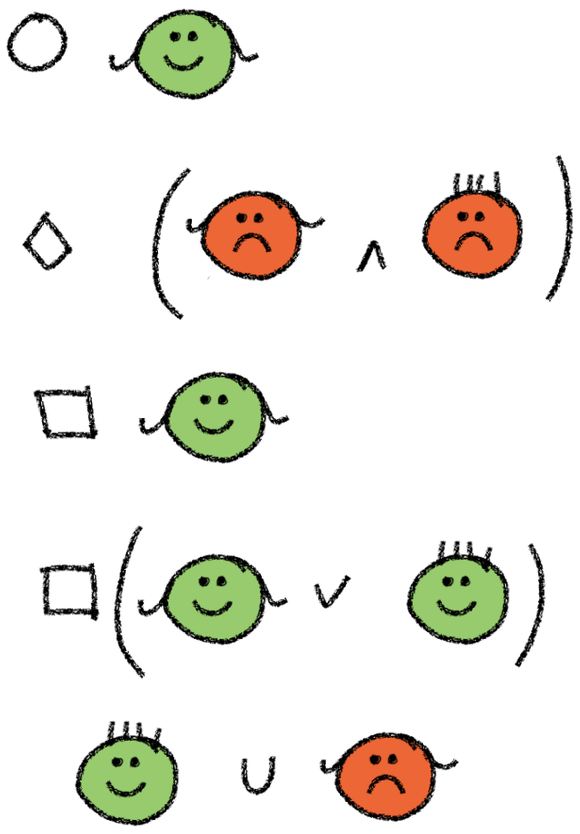


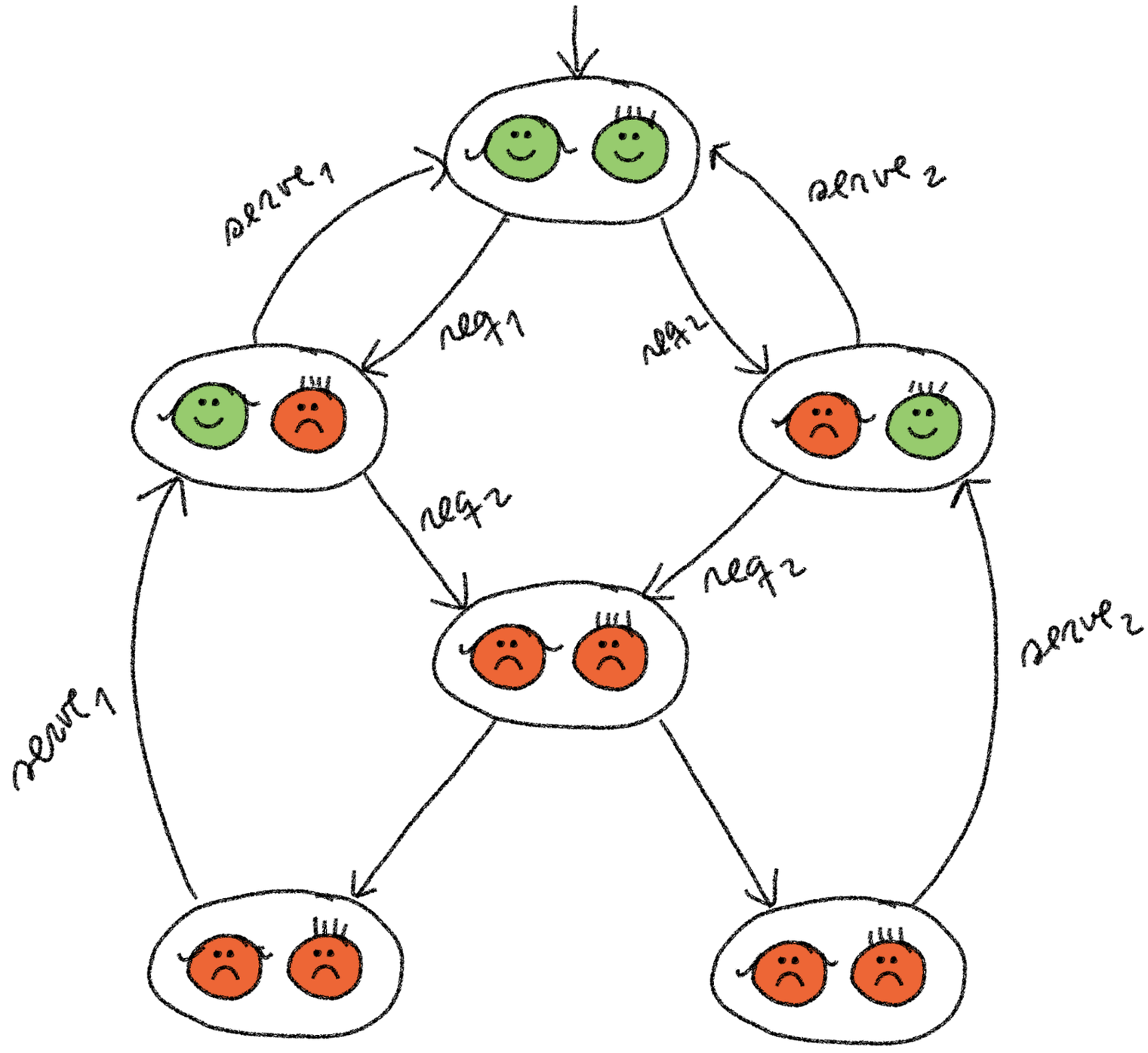
Intuition of temporal operators





Some examples



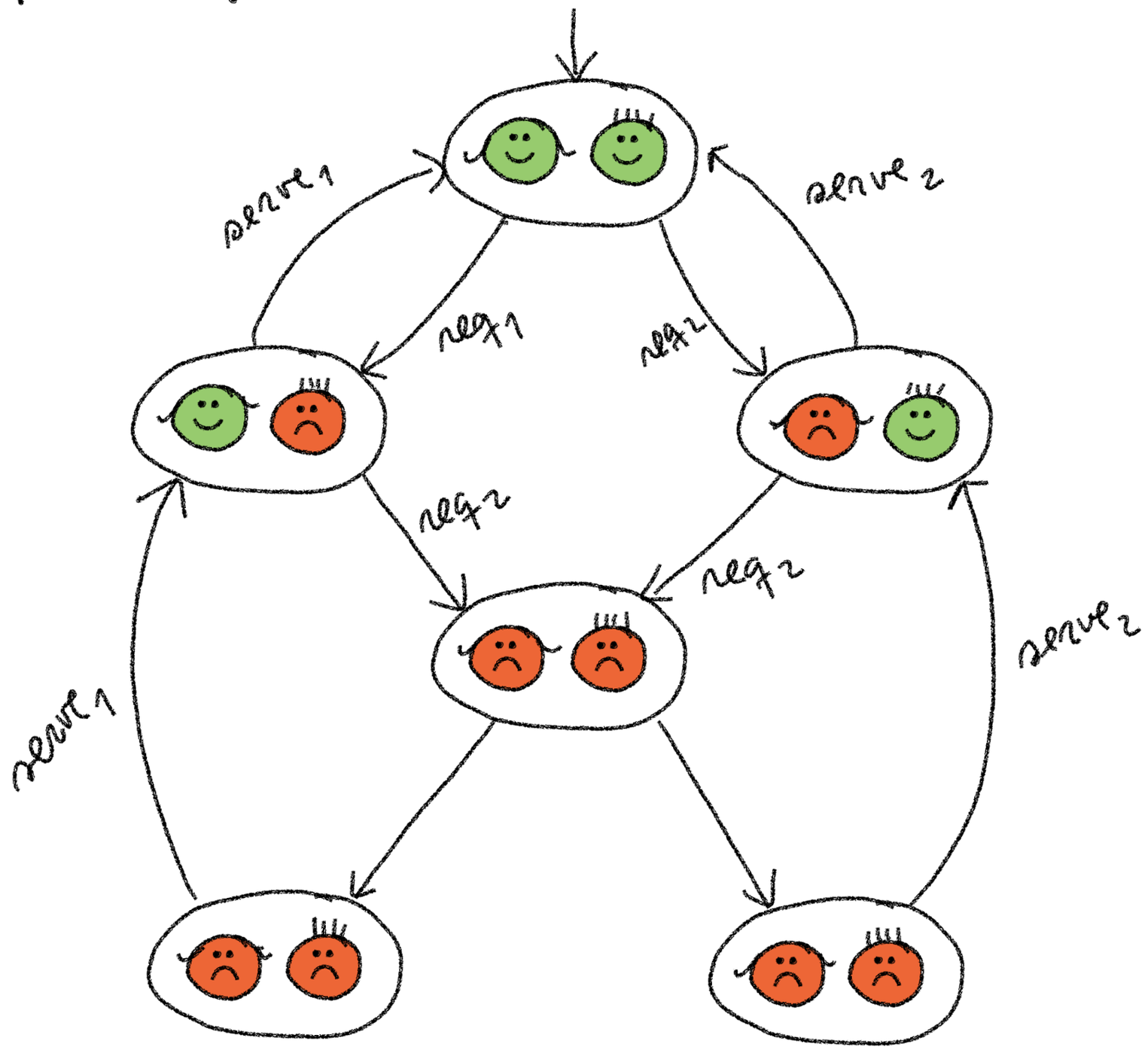


DEMO?
PRISM

Some examples

Find formulas satisfied only by the initial state ...

↓



Witnesses and Counterexamples

A **witness** for a formula is an explanation of why it holds.

A **counterexample** is an explanation of why it does not hold.

NOTE: yes, model checking has explainability built-in :)

In LTL…

Witnesses and Counterexamples

A **witness** for a formula is an explanation of why it holds.

A **counterexample** is an explanation of why it does not hold.

NOTE: yes, model checking has explainability built-in :)

In LTL…

A **trivial witness** is the entire transitions system (sorry).

A **counterexample** is just a so-called **lasso**:

- a finite initial path fragment ending in state s
- a loop starting and ending at s

For some formulas it can be simpler (see next slide).

Some examples

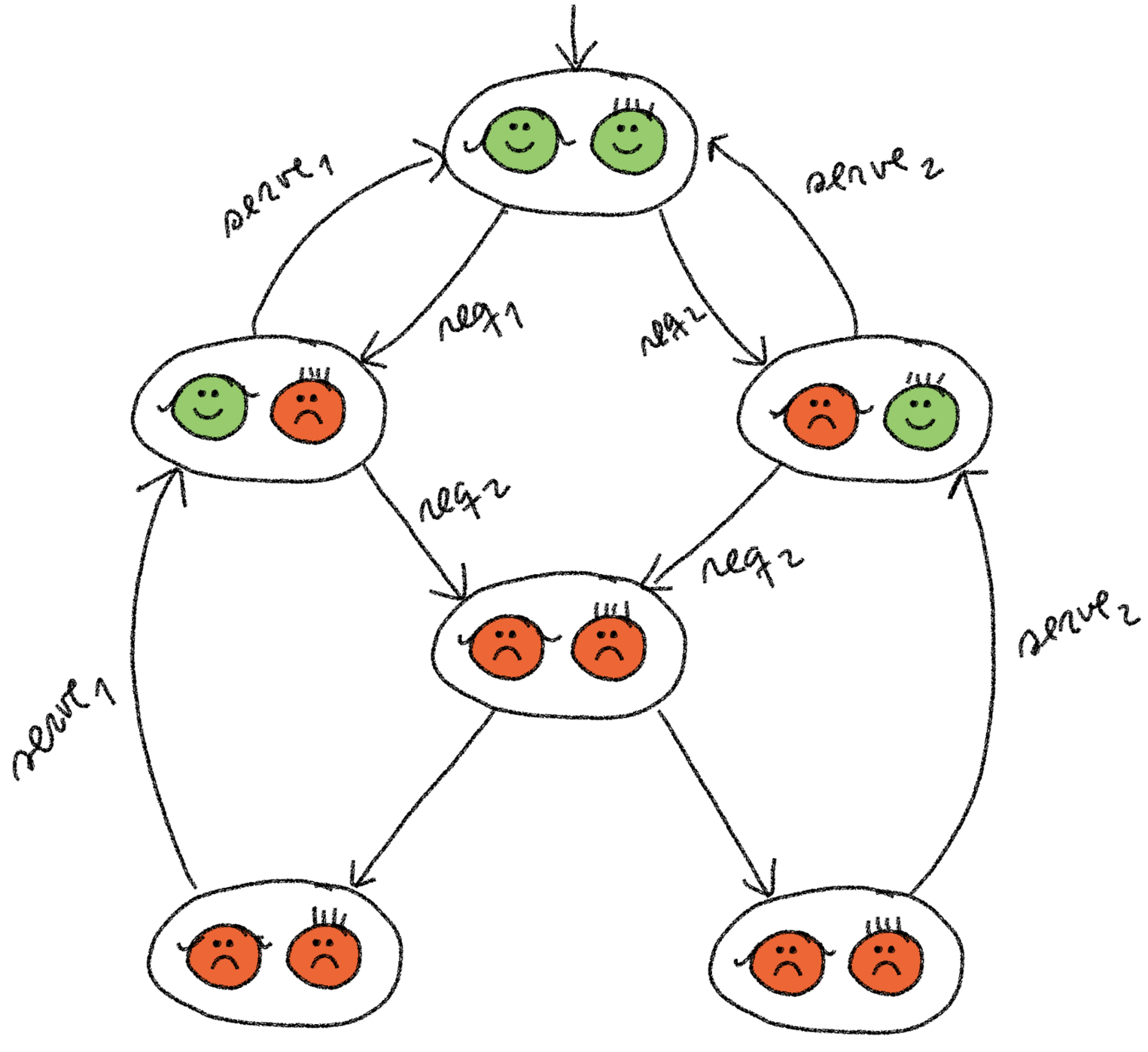
Fill the formulas so they do not hold and find corresponding counterexamples (lassos)

X _

F _

G _

_ U _



Counterexamples for basic LTL formulas

i.e., assume the below formulas are violated —> which lassos could cause it?

Typical Patterns of Formulas

"inv is an invariant"

$$\Box\, inv$$

"every request if followed by a response"

$$\Box(request \rightarrow \Diamond response)$$

"infinitely often p holds" (or "always eventually p")

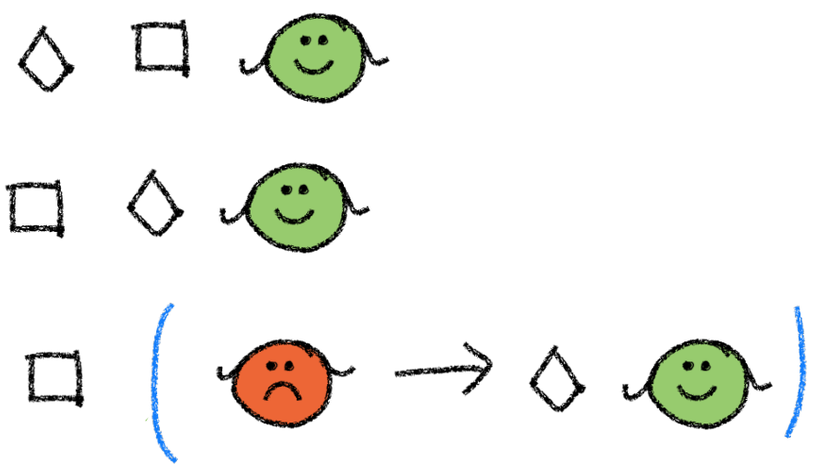
$$\Box\Diamond p$$

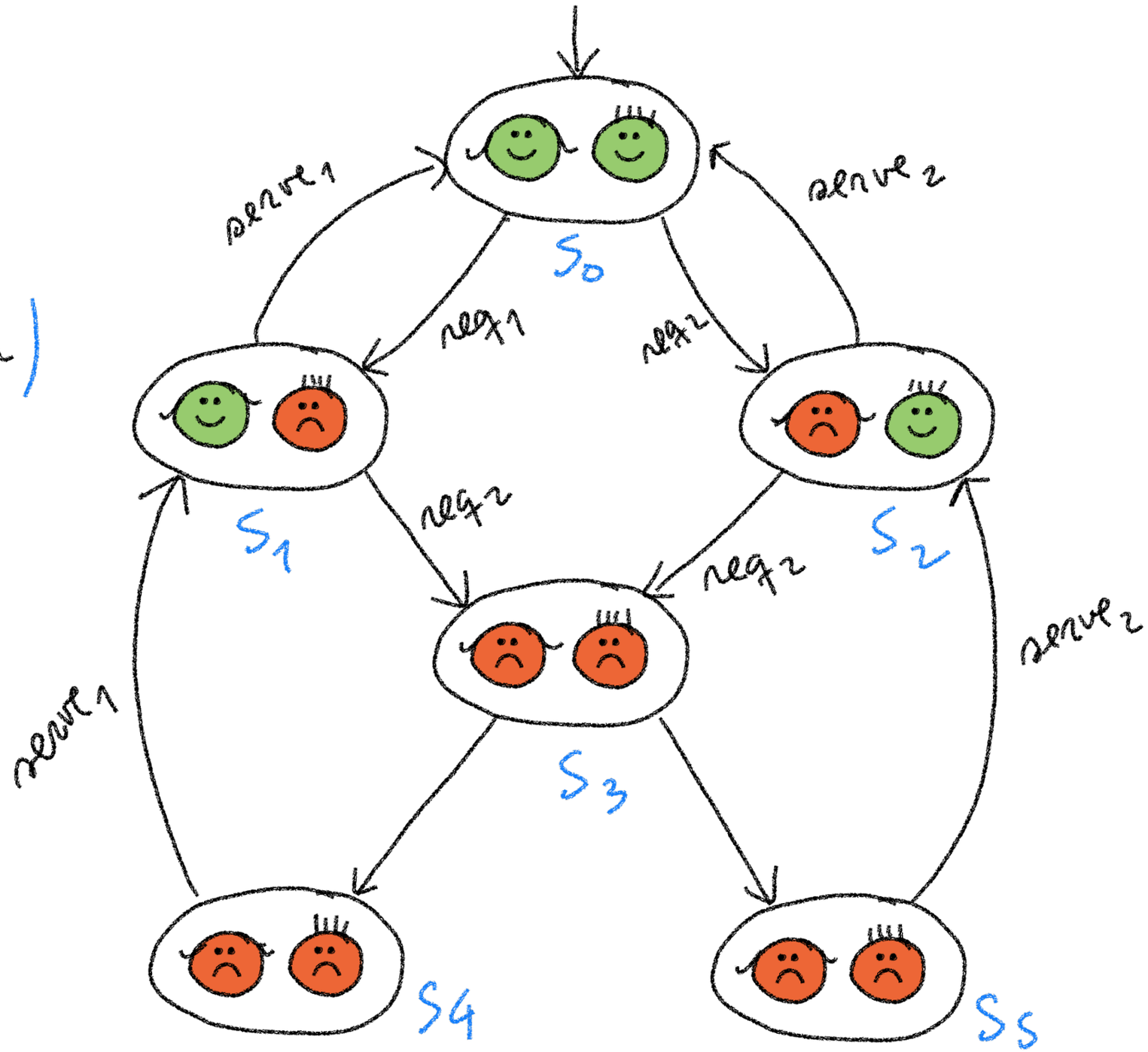
"q is persistent" (or "eventually forever q")

$$\Diamond\Box q$$

NOTE: A collection of property patterns is available at https://matthewbdwyer.github.io/psp/

More examples





NOTE: In PRISM, logical operators have precedence over temporal ones

Formal semantics…

Some notation: paths, prefixes, states

For a state s we define the **set of all paths** starting from s, as the set of all maximal executions starting from s.

$$Paths(s) = \{s_0, s_1, s_2, \ldots \mid s_0 = s \text{ and } s_i \rightarrow s_{i+1}\}$$

For a path $\pi = s_0, s_1, s_2, \ldots$ we define the (i-1)-th state by

$$\pi[i] = s_i$$

We can also define the prefix starting at the (i-1)-th state

$$\pi[i\,..\,] = s_i, s_{i+1}, \ldots$$

LTL - formal semantics

A state satisfies a formula if all paths starting from s satisfy the formula

$$s \vDash \phi \quad \text{iff} \quad \forall \pi \in Paths(s) \,.\, \pi \vDash \phi$$

Path satisfaction semantics of propositional fragment

$\pi \vDash_{LTL} true$ (holds always)

$\pi \vDash_{LTL} p$ iff $p \in L(\pi[0])$

$\pi \vDash_{LTL} \neg\phi$ iff $\pi \nvDash_{LTL} \phi$

$\pi \vDash_{LTL} \phi_1 \wedge \phi_2$ iff $\pi \vDash_{LTL} \phi_1$ and $\pi \vDash_{LTL} \phi_2$

$\pi \vDash_{LTL} \phi_1 \vee \phi_2$ iff $\pi \vDash_{LTL} \phi_1$ or $\pi \vDash_{LTL} \phi_2$

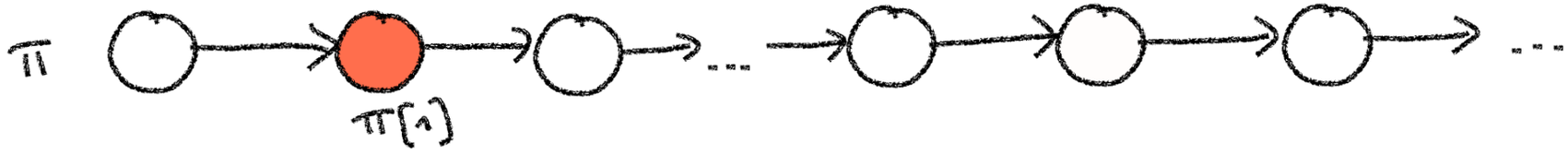
Etc.

Semantics of LTL

A state s satisfies a formula if all executions from s satisfies the formula.

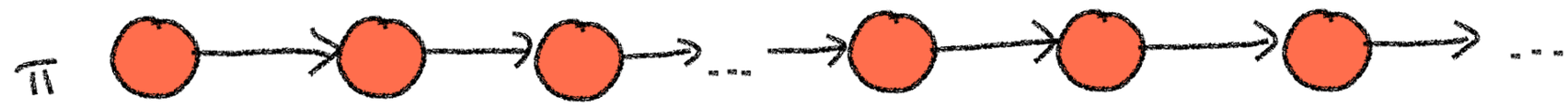
An execution satisfies a formula according to the following satisfaction relation:

$\pi \vDash_{LTL} \bigcirc \phi$ iff $\pi[1..] \vDash_{LTL} \phi$

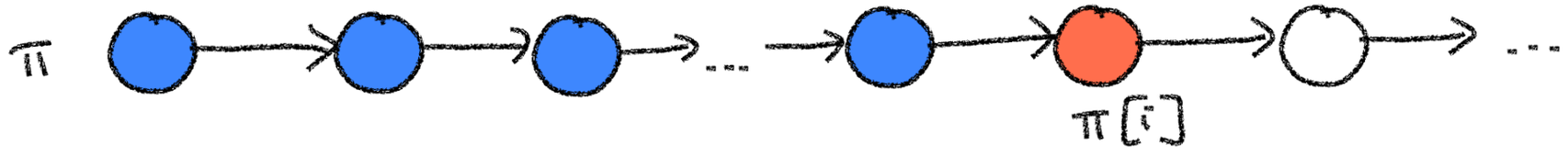


$\pi \vDash_{LTL} \Diamond \phi$ iff $\exists i \in \mathbb{N} . \pi[i..] \vDash_{LTL} \phi$

$\pi \vDash_{LTL} \Box \phi$



$\pi \vDash_{LTL} \phi_1 \mathsf{U} \phi_2$

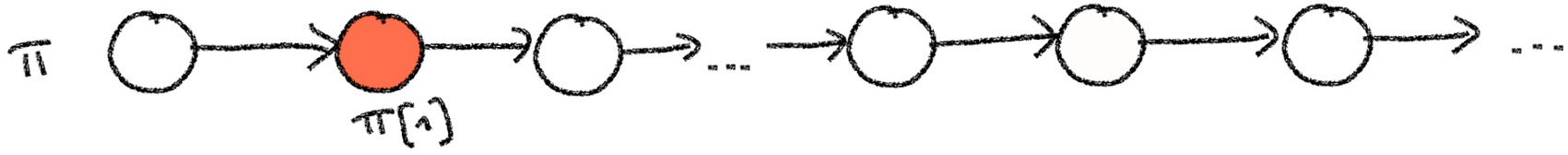


Semantics of LTL

A state s satisfies a formula if all executions from s satisfies the formula.

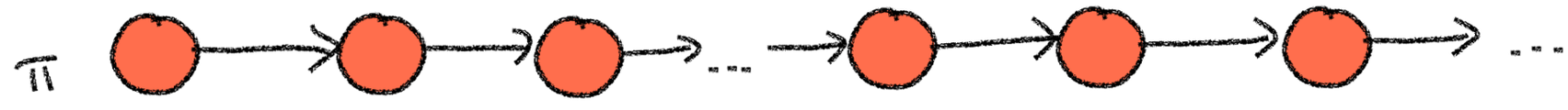
An execution satisfies a formula according to the following satisfaction relation:

$\pi \vDash_{LTL} \bigcirc \phi$ iff $\pi[1..] \vDash_{LTL} \phi$

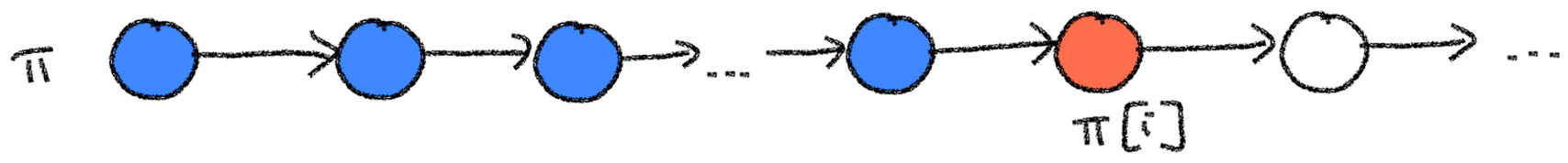


$\pi \vDash_{LTL} \Diamond \phi$ iff $\exists i \in \mathbb{N} . \pi[i..] \vDash_{LTL} \phi$

$\pi \vDash_{LTL} \Box \phi$ iff $\forall i \in \mathbb{N} . \pi[i..] \vDash_{LTL} \phi$



$\pi \vDash_{LTL} \phi_1 \mathsf{U} \phi_2$ iff $\exists i \in \mathbb{N} . \pi[i..] \vDash_{LTL} \phi_2 \wedge \forall 0 \leq j < i . \pi[j..] \vDash_{LTL} \phi_1$



Satisfaction Sets

We define the satisfaction set of an LTL or CTL formula as the set of states that satisfy the formula

$$sat(\phi) = \{s \mid s \vDash \phi\}$$

Semantics over a TS

We say that a transition system satisfies a formula if all its initial states satisfy the formula:

$$T \models \phi \text{ iff } \forall s \in I \,.\, s \models \phi$$

or, equivalently

$$T \models \phi \text{ iff } I \subseteq sat(\phi)$$

A state satisfies a formula if all paths starting from s satisfy the formula

$$s \models \phi \text{ iff } \forall \pi \in Paths(s) \,.\, \pi \models \phi$$

Equivalences…

Equivalences

Two formulas are equivalent iff their validity (hold/not-holds) is **the same for all transitions systems**, i.e. for all transition systems T we have:

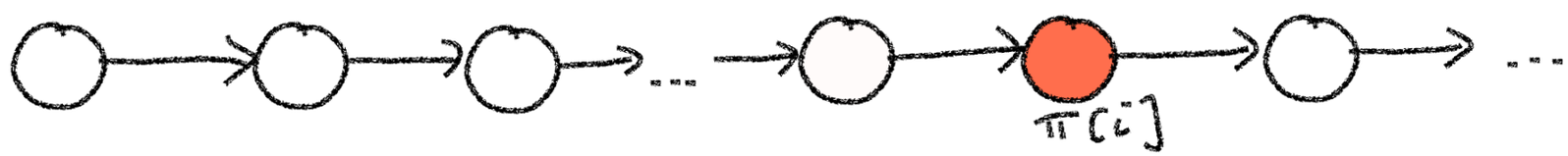
$$T \vDash \phi_1 \qquad \text{iff} \qquad T \vDash \phi_2$$

“Eventually” is a case of “Until”

A simple equivalence is allows use to define “eventually” with “until”:

$$\Diamond\phi \equiv \mathit{true}\ \mathsf{U}\ \phi$$

$\pi \vDash \Diamond\phi_2$ iff $\exists i \in \mathbb{N} . \pi[i] \vDash \phi_2$



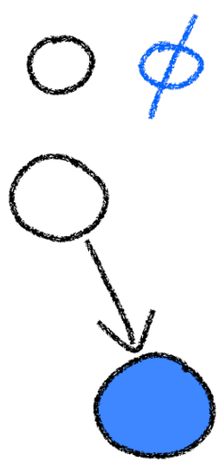
$\pi \vDash \phi_1 \mathsf{U} \phi_2$ iff $\exists i \in \mathbb{N} . \pi[i] \vDash \phi_2 \wedge \forall 0 \leq j < i . \pi[j] \vDash \phi_1$

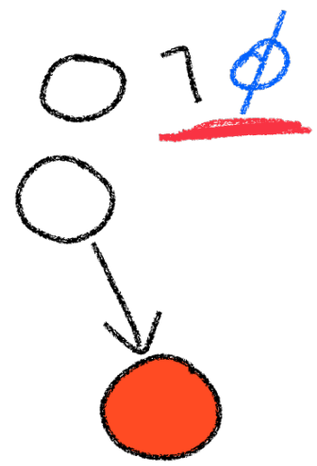
Dualities for “next”

Another example of simple equivalence is

$$\bigcirc\phi \equiv \neg\bigcirc\neg\phi$$

Easy to see graphically





Dualities for “eventually” and “always”

Does the following hold?

$$\Box \phi \equiv \neg \Diamond \neg \phi$$

More Distribution Laws?

Does this hold?

$$\Diamond(\phi_1 \wedge \phi_2) \equiv (\Diamond\phi_1) \wedge (\Diamond\phi_2)$$

More Distribution Laws?

Does this hold?

$$\Diamond(\phi_1 \wedge \phi_2) \equiv (\Diamond\phi_1) \wedge (\Diamond\phi_2)$$

How to argue?

- Important for mandatory assignments
- Equivalence/implication holds —> demonstrate formally
- Equivalence/implication doesn’t hold —> show a counter-example

More laws

duality law

$$\neg \bigcirc \varphi \equiv \bigcirc \neg \varphi$$
$$\neg \Diamond \varphi \equiv \Box \neg \varphi$$
$$\neg \Box \varphi \equiv \Diamond \neg \varphi$$

idempotency law

$$\Diamond \Diamond \varphi \equiv \Diamond \varphi$$
$$\Box \Box \varphi \equiv \Box \varphi$$
$$\varphi \,\mathsf{U}\, (\varphi \,\mathsf{U}\, \psi) \equiv \varphi \,\mathsf{U}\, \psi$$
$$(\varphi \,\mathsf{U}\, \psi) \,\mathsf{U}\, \psi \equiv \varphi \,\mathsf{U}\, \psi$$

absorption law

$$\Diamond \Box \Diamond \varphi \equiv \Box \Diamond \varphi$$
$$\Box \Diamond \Box \varphi \equiv \Diamond \Box \varphi$$

expansion law

$$\varphi \,\mathsf{U}\, \psi \equiv \psi \vee (\varphi \wedge \bigcirc (\varphi \,\mathsf{U}\, \psi))$$
$$\Diamond \psi \equiv \psi \vee \bigcirc \Diamond \psi$$
$$\Box \psi \equiv \psi \wedge \bigcirc \Box \psi$$

distributive law

$$\bigcirc (\varphi \,\mathsf{U}\, \psi) \equiv (\bigcirc \varphi) \,\mathsf{U}\, (\bigcirc \psi)$$
$$\Diamond (\varphi \vee \psi) \equiv \Diamond \varphi \vee \Diamond \psi$$
$$\Box (\varphi \wedge \psi) \equiv \Box \varphi \wedge \Box \psi$$

Figure 5.7: Some equivalence rules for LTL.

See MC book

Expansion laws

Can be thought of recursively explaining the meaning of formulas:

$$\Box \phi \equiv \phi \wedge \bigcirc \Box \phi$$

$$\Diamond \phi \equiv \phi \vee \bigcirc \Diamond \phi$$

$$\phi_1 \mathsf{U} \phi_2 \equiv \phi_2 \vee (\phi_1 \wedge \bigcirc \phi_1 \mathsf{U} \phi_2)$$

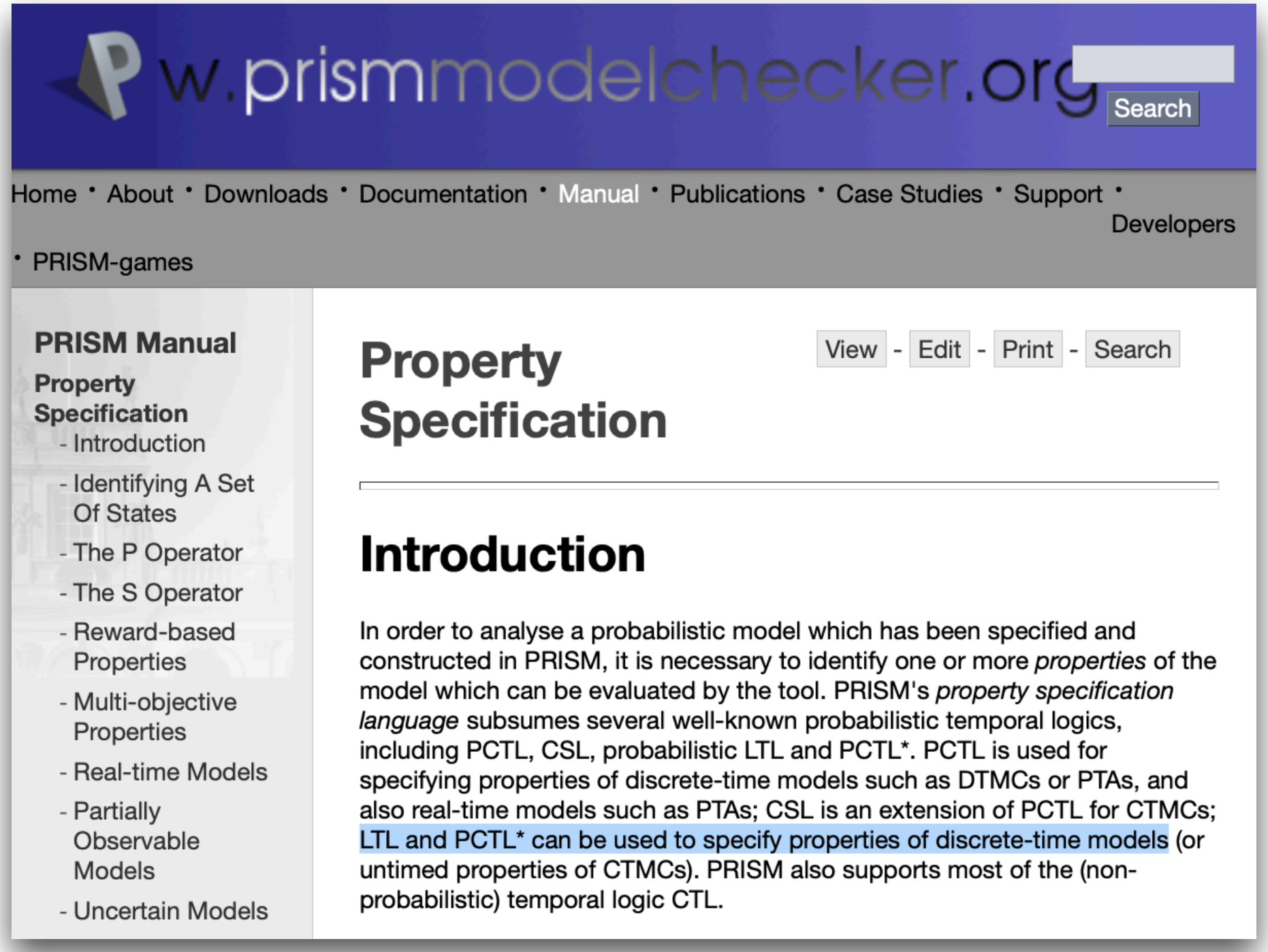
LTL - minimal syntax

Can we get rid of some operators in LTL?

$$\phi ::= true \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc\phi \mid \Diamond\phi \mid \Box\phi \mid \phi_1 \mathsf{U} \phi_2$$

LTL in PRISM

LTL in PRISM



See https://www.prismmodelchecker.org/manual/PropertySpecification/AllOnOnePage

BONUS: Recall on SAT…

Propositional logic: satisfiability

SAT problem

Given a propositional formula φ. Is there a valuation v such that $v(\varphi) = \mathrm{T}$?

Propositional logic: satisfiability

SAT problem

Given a propositional formula φ. Is there a valuation v such that $v(\varphi) = \mathrm{T}$?

Theorem Formulas φ is valid iff and $\neg\varphi$ is unsatisfiable

Propositional logic: satisfiability

SAT problem

Given a propositional formula φ. Is there a valuation v such that $v(\varphi) = \mathrm{T}$?

Theorem Formulas φ is valid iff and $\neg\varphi$ is unsatisfiable

Theorem Satisfiability and validity are decidable

Brute-force approach

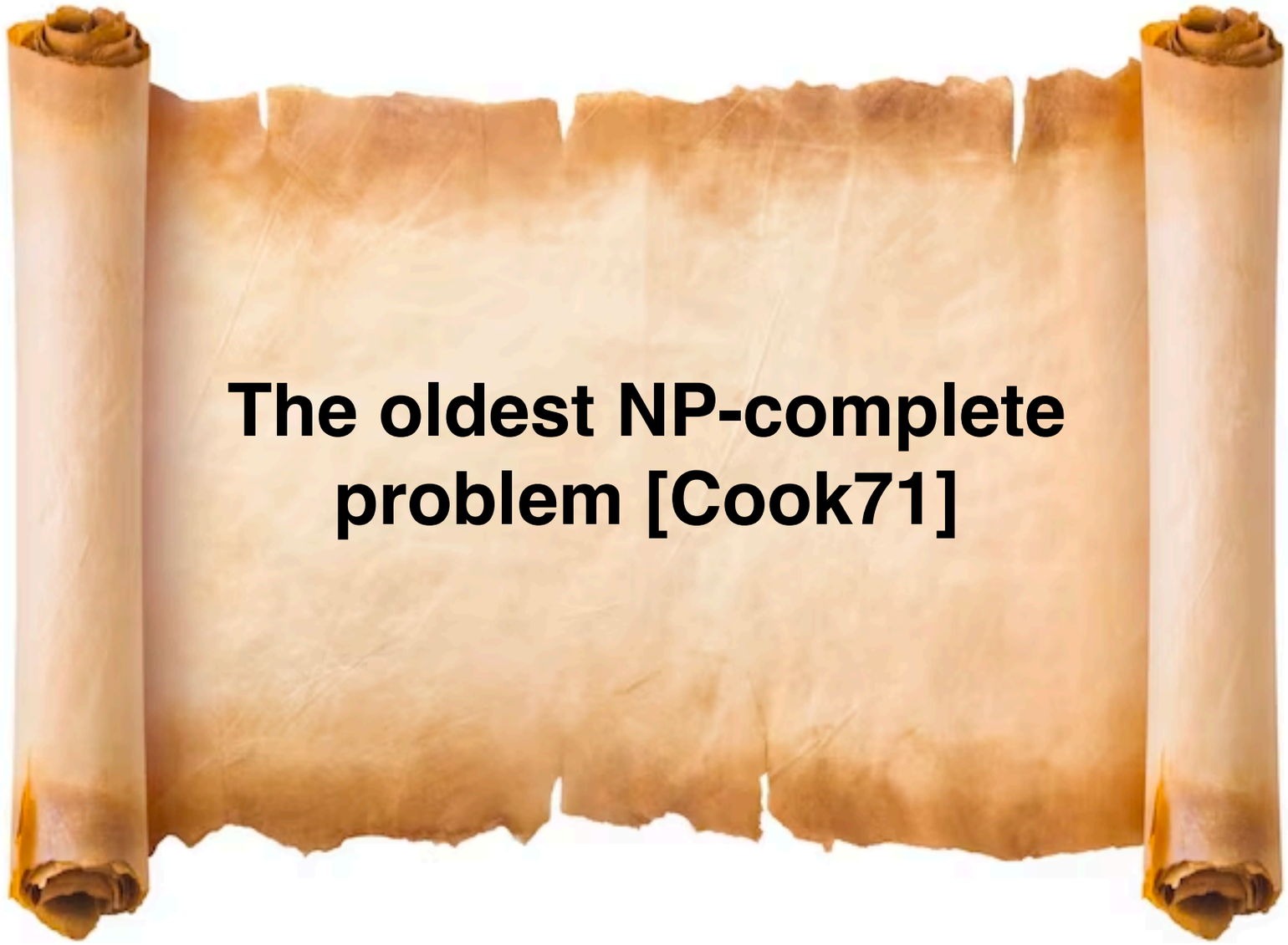
Can use truth tables to check satisfiability!

Create a truth table for the given formula φ. If the column for φ contains $\mathtt{T}$, then φ is **satisfiable**!

p	q	z	$\neg p \wedge (q \vee \neg p) \wedge (z \vee \neg q)$
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	

Obviously, this is not what SAT solvers do :)

SAT to solve ‘em all!



[Cook71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971

So… what does it give us?

SAT to solve ‘em all!

NP problem

- Decision problem (returns “yes”/“no”)
- Decidable (in a finite number of operations)
- Finding a solution is difficult (in the worst case, **exponential time**)
- Checking a solution (*does the valuation satisfy the formula?*) is easy (**polynomial time**)

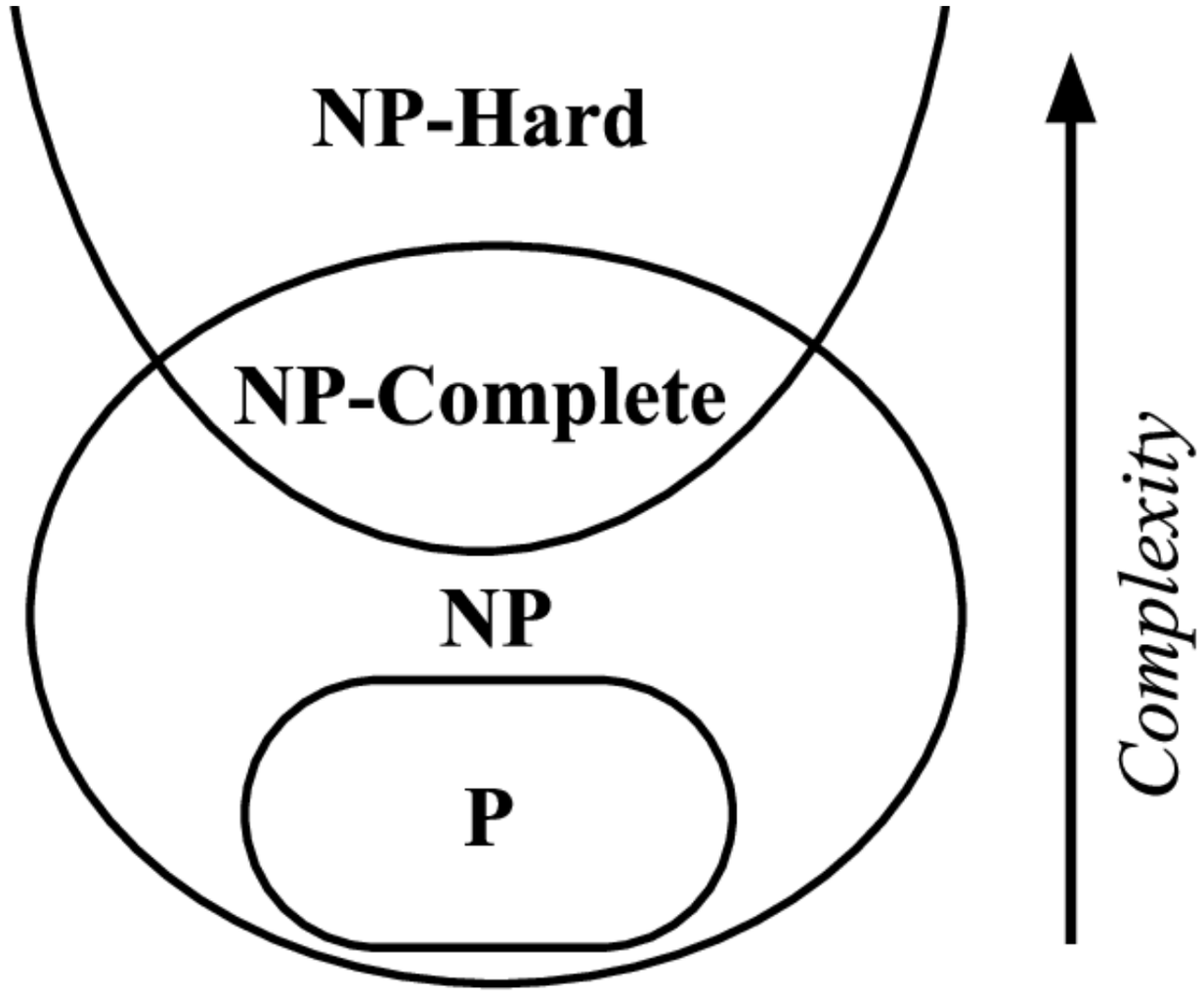
NP = the set of all problems that are solvable in polynomial time by a “non-deterministic Turing machine”

SAT to solve ‘em all!

NP-complete problems are the “most difficult” problems in NP.

All problems in NP can be translated into NP-complete problems in polynomial time.

⇒ NP/NP-complete problems **can be solved by encoding** them **into SAT**!



SAT to solve ‘em all!

⇒ Building a “very fast” SAT solver could be used for solving lots of other not-so-easy problems!

In theory: wishful thinking, NP-problems are known to take exponential time in the worst case.

In practice: modern SAT solvers are very fast most of the time!

BONUS: LTL satisfiability

We will mainly focus on the model checking problem. Given M and Phi decide

$$M \models \Phi ?$$

But will also consider the SAT problem: does there exist a model M that satisfies Phi?

$$\exists M \,.\, M \models \Phi ?$$

Why would this problem be interesting in practice?

What about SAT for LTL?

SAT problem

Given a ~~propositional~~ LTL formula φ. Is there a valuation v such that $v(\varphi) = \mathrm{T}$?

Theorem Formulas φ is unsatisfiable iff and $\neg\varphi$ is valid

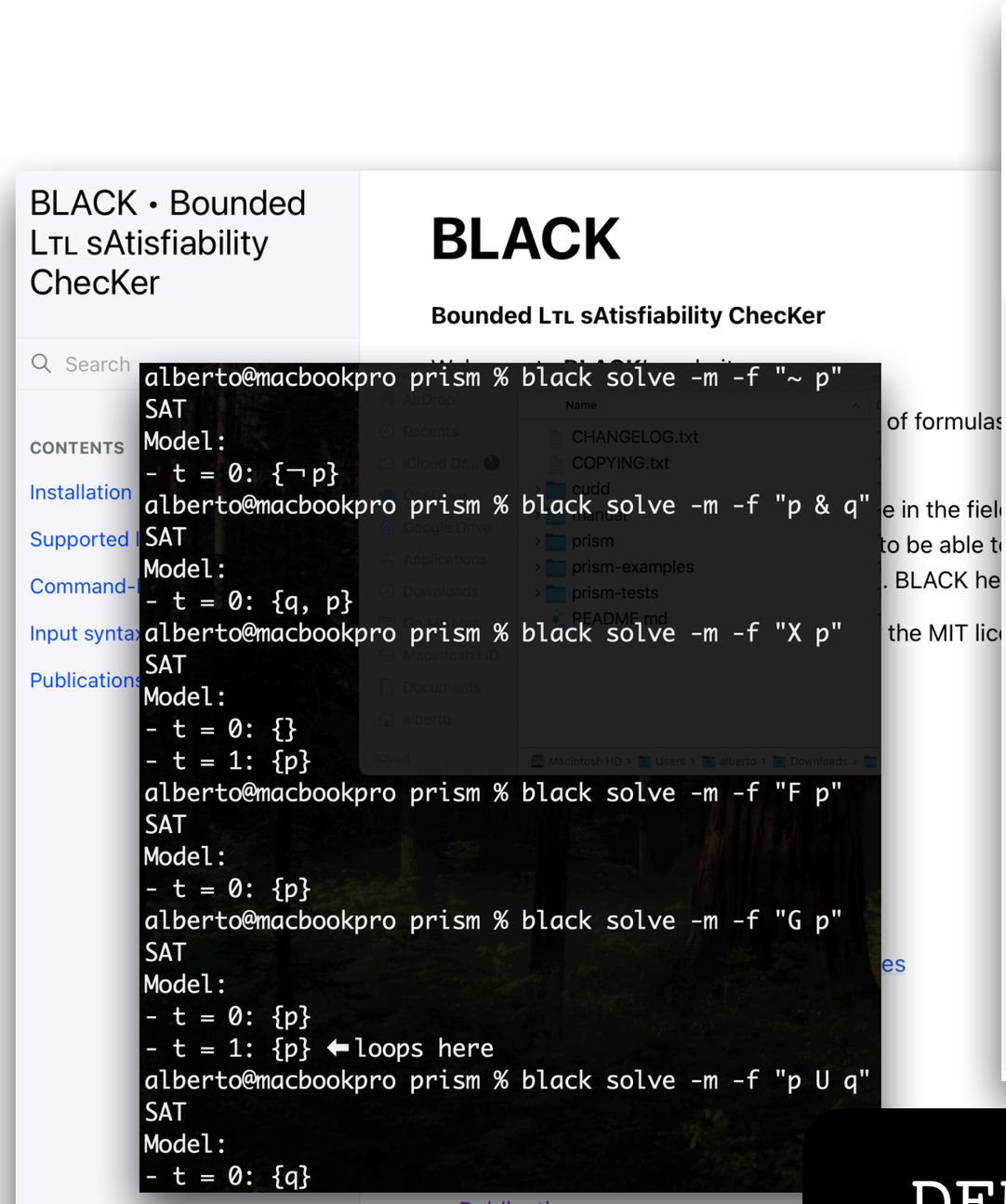
Theorem Satisfiability and validity are decidable

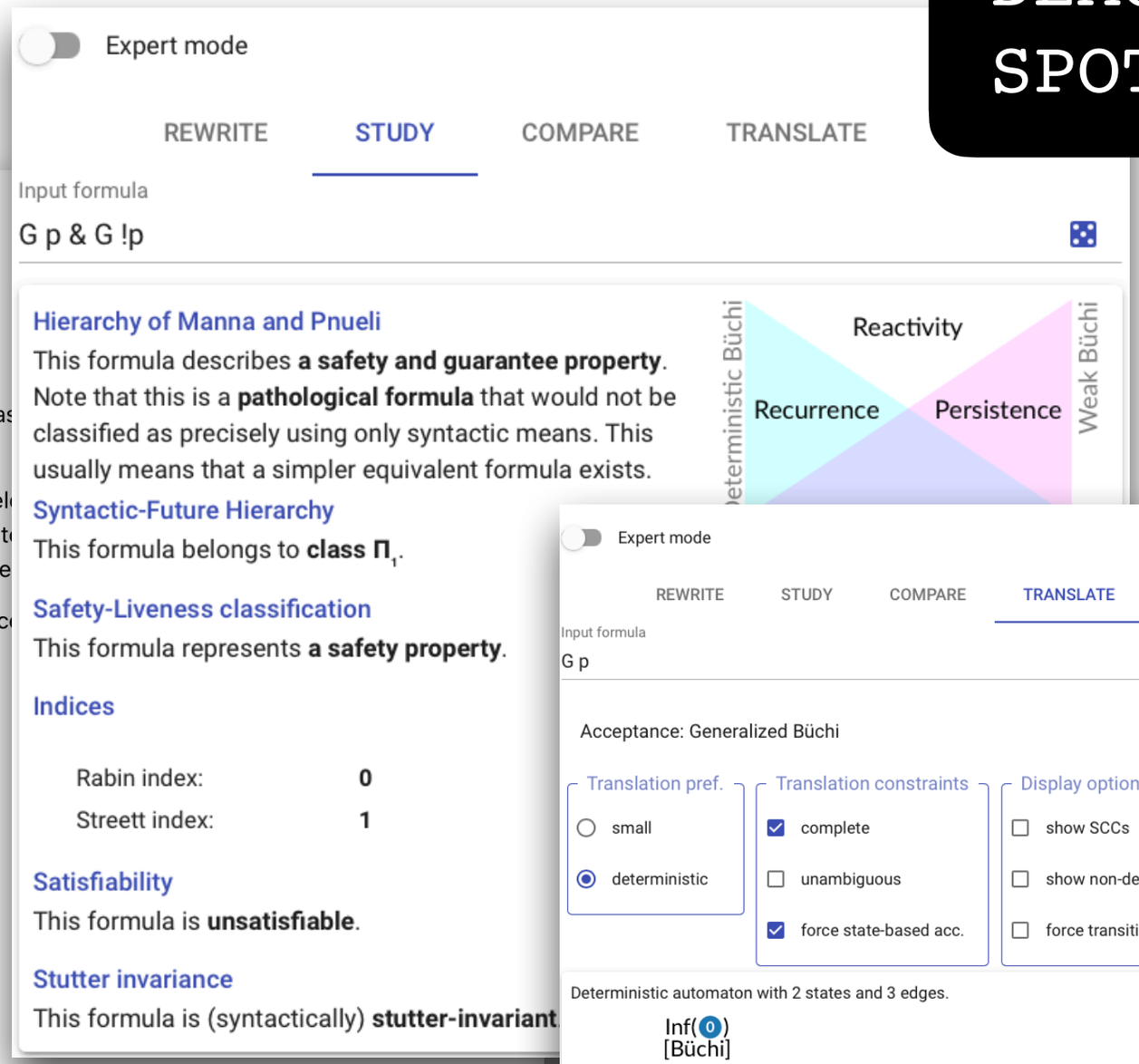
If curious, see 5.2.2

What about SAT for LTL?

LTL model checkers and tools like black and SPOT can save the day

DEMO?
SPOT?





DEMO?BLACK

https://www.black-sat.org/

Will find one trace (hence one TS) contained in the traces of phi

https://spot.lre.epita.fr/app/

Will find the automata equivalent to phi
(i.e. accept the same set of traces)

SUPER BONUS: Model checking LTL algorithms?

Algorithms in a nutshell

MC algorithms for other logics will be covered later in the course.

Two popular approaches for LTL:

(1) Automata-based

(2) SAT-based (aka bounded model checking)

Automata-based LTL

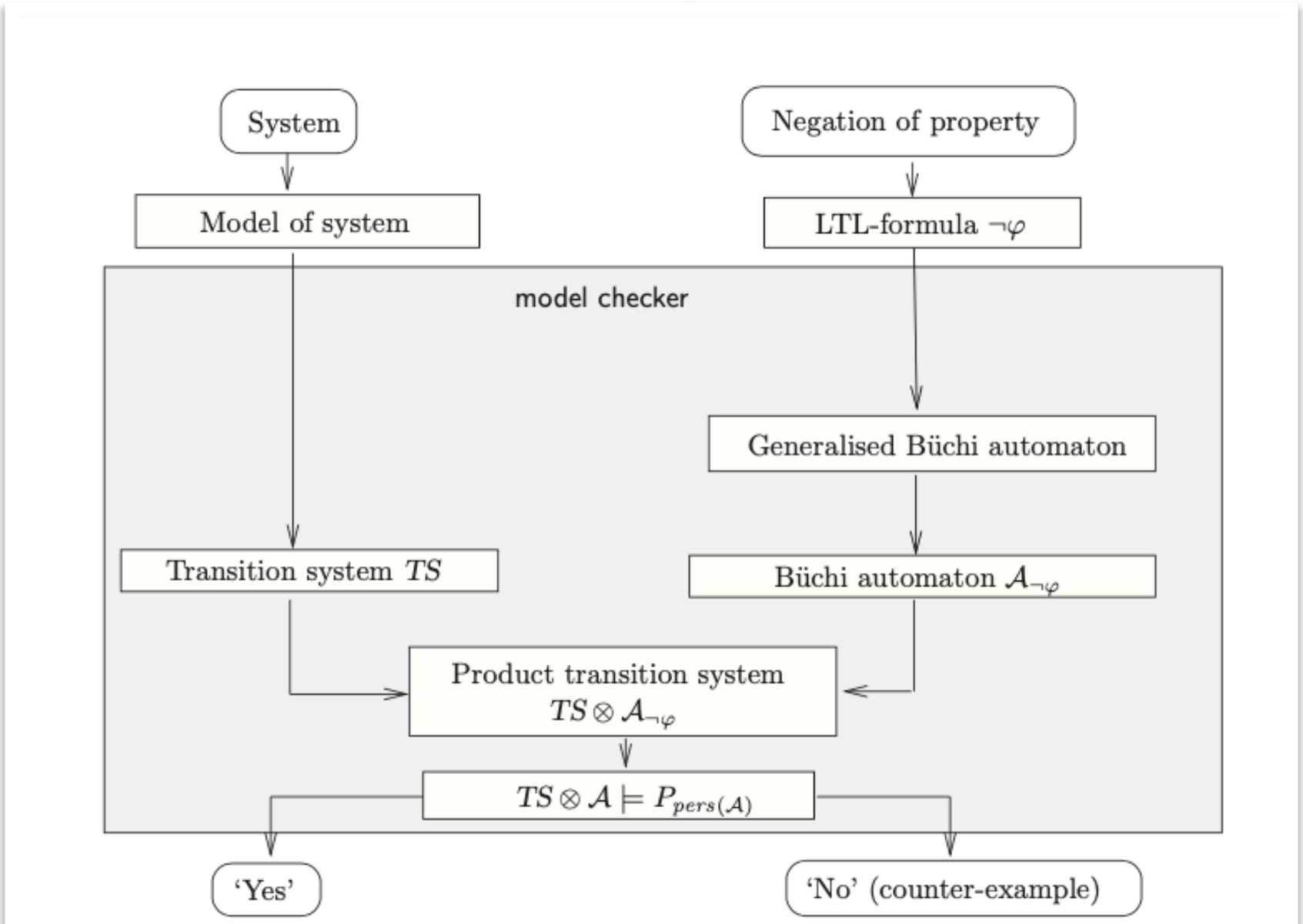


Figure 5.16: Overview of LTL model checking.

See MC book chapter 5.2

Bounded Model Checking

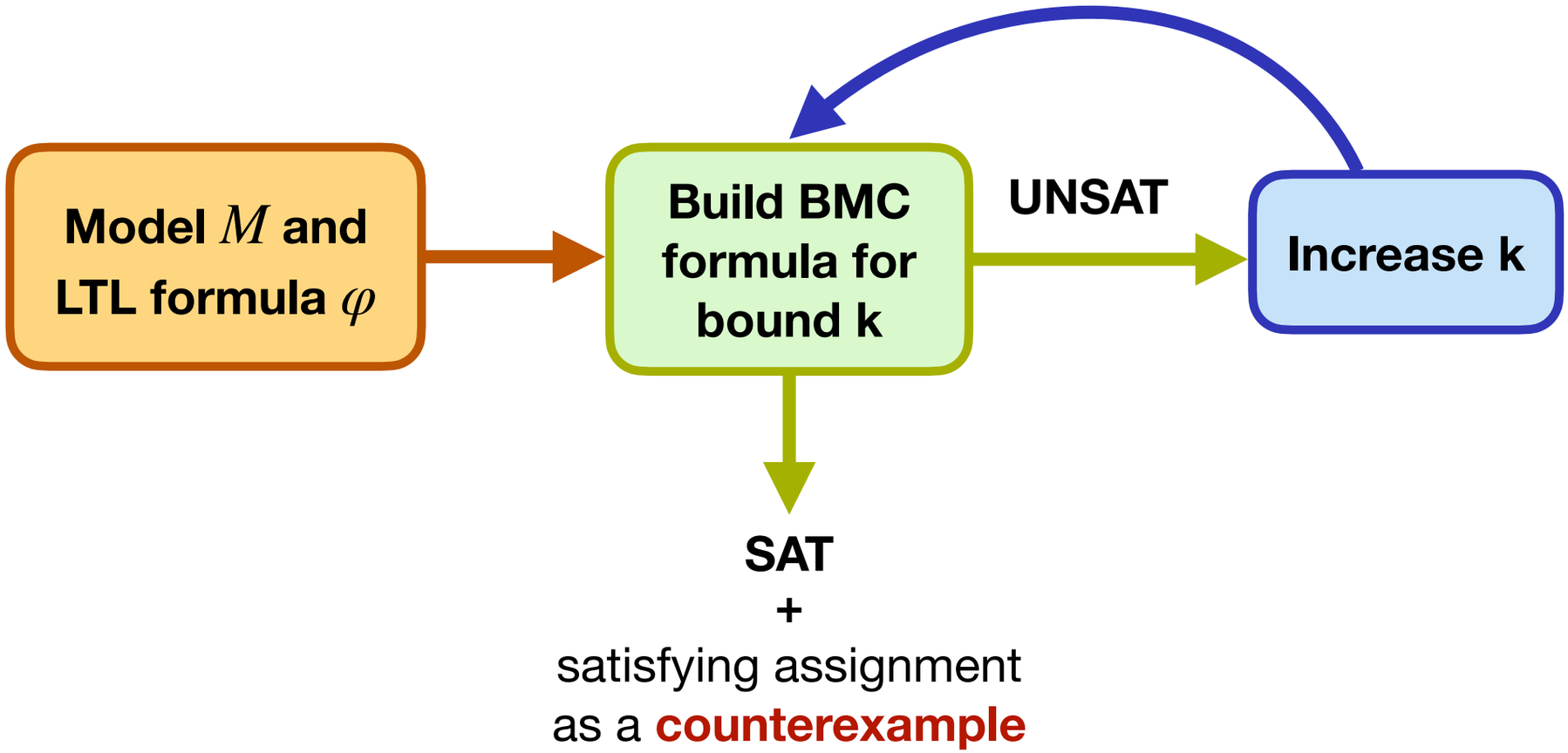
Proposed by Biere, Clarke et al. [Biere et al., 1999]

Idea: Search for a **counterexample in executions** (paths) whose length is **≤ k** for some integer **k**; this example is called a **bounded witness** of a temporal property (*an initialised infinite path, i.e., starting with one of the initial sates, in which the property holds, and which can be represented by a finite path of length k*)

- BMC exploits the power of SAT/SMT
- Generates a formula F corresponding to an unfolding of the given model M (and property φ) for a given depth bound k
- F is SAT iff φ can be **refuted** on M by means of a **counterexample of length k**

Bounded Model Checking

Naive BMC workflow



- BMC finds counterexamples of minimal, bounded length (**helps understanding counterexamples**)
- **Main purpose:** bug/error finding in the design
- Works with the symbolic encoding of paths, doesn't require additional constructions (e.g., automata for LTL)

Closing…

Key points of this lecture

Temporal Logics as an extension of propositional logics to reason about transition systems.

Linear-time Temporal Logic (LTL) as a logic to about reason traces.

Grammars for LTL.

Formal semantics of LTL: satisfaction relation for paths.

Derived operators and equivalences between LTL formulas.

How to write LTL formulas in PRISM.

Exercises

Exercise 02.1

Determine if the transition system satisfies the below formulas

(1) $\Diamond y$

(2) $\Box y$

(3) $\Box \Diamond y$

(4) $\Diamond g$

(5) $\Diamond b$

(6) $\Box y$

(7) $\Box \neg g$

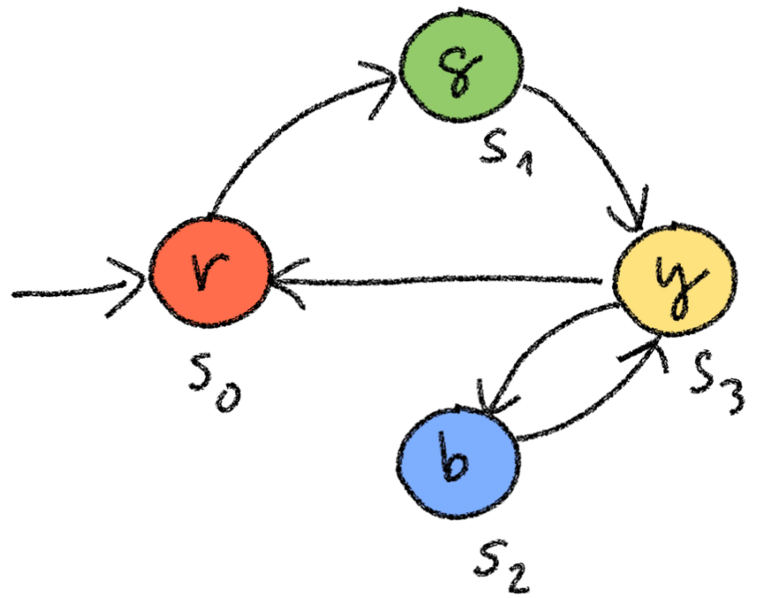
(8) $(b \mathsf{U} \neg b)$

(9) $\bigcirc (g \vee r)$

(10) $(g \mathsf{U} \Box (b \vee y))$

(11) $(g \mathsf{U} (y \mathsf{U} r))$

(12) $(\neg b \mathsf{U} b)$



NOTE: Reason using the semantics of LTL. If you are in doubt you can double-check with PRISM.

Exercise 02.2

Consider the following statements about CTL formulas:

(1) $(\Box\phi_1) \vee (\Box\phi_2)$ implies $\Box(\phi_1 \vee \phi_2)$

(2) $\Box(\phi_1 \vee \phi_2)$ implies $(\Box\phi_1) \vee (\Box\phi_2)$

For each statement determine if it holds or not.

If the statement does not hold, provide a counterexample (a transition system that satisfies the formula on the left but not the one on the right).

If the statement holds, provide an explanation (similar to the ones in the book or slides).

Let p, q be atomic propositions and consider the following formulas

(1) FX p
(2) (F p) ∧ (¬p) ∧ (¬X p) ∧ (¬X X p)
(3) (GF p) ∧ (GF ¬p)
(4) (GF p) ∧ (G¬F p)
(5) (GF p) ∧ (¬GF p)

For each formula… is it satisfiable?

If yes, provide model (a transition system). Make it as small as possible.

If the formula is not satisfiable explain why.

Exercise 02.4: hands on Z3

1. Get Z3 for Python
   - easy in Google colab, just add these lines…
   - …and start Z3-coding!
   - Alternatively, VS Code, terminal, etc.
2. Run tutorial https://ericpony.github.io/z3py-tutorial/guide-examples.htm
3. Solve the SAT and validity questions for propositional formulas from the slides using Z3.

```
!pip install z3-solver
from z3 import *
```

PS. Z3 will be needed later in the course (probabilistic part), so why not start using it :)

Solutions

Exercise 02.1

Determine if the transition system satisfies the below formulas

(1) $\Diamond y$

(2) $\Box y$

(3) $\Box \Diamond y$

(4) $\Diamond g$

(5) $\Diamond b$

(6) $\Box y$

(7) $\Box \neg g$

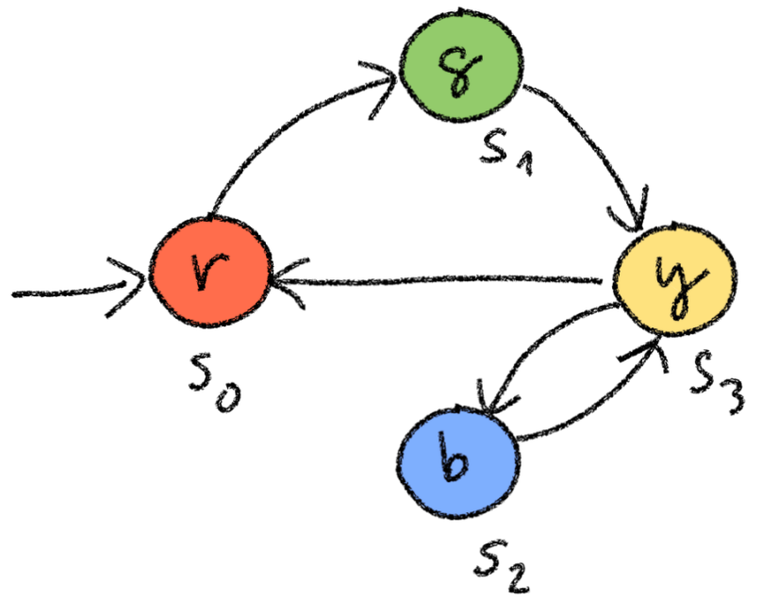
(8) $(b \mathsf{U} \neg b)$

(9) $\bigcirc (g \vee r)$

(10) $(g \mathsf{U} \Box (b \vee y))$

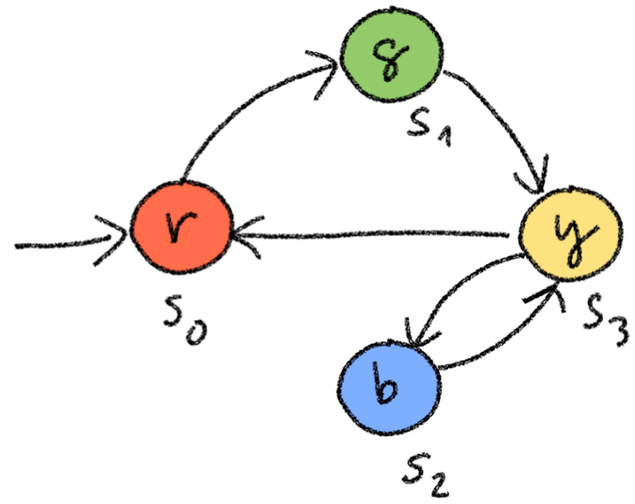
(11) $(g \mathsf{U} (y \mathsf{U} r))$

(12) $(\neg b \mathsf{U} b)$



NOTE: Reason using the semantics of LTL. If you are in doubt you can double-check with PRISM.

Exercise 02.1



Short answers below, but the p

(1) $\Diamond y$ **YES. All traces hit s3 in the 3rd state of the trace.**

(2) $\Box y$ **NO. Any trace would be a counterexample, as y is not true in the initial state of any trace.**

(3) $\Box \Diamond y$ **YES. No trace can avoid going through s3 infinitely often.**

(4) $\Diamond g$ **YES. All traces hit s1 in the 2nd state.**

(5) $\Diamond b$ **NO. A counterexample is the execution (s0,s1,s3)$^\omega$. Seen as a trace, it never hits b.**

(6) $\Box y$ **NO. Same as (2) :)**

(7) $\Box \neg g$ **NO. Any trace would be a counterexample, as g is true in the second state of any trace.**

(8) $(b \cup \neg b)$ **YES. Any trace starts with s0, where b does not hold, hence the RHS of the until is satisfied.**

(9) $\bigcirc(g \vee r)$ **YES. All traces hit s1 as their second state, where g holds.**

(10) $(g \cup \Box(b \vee y))$ **NO.**

(11) $(g \cup (y \cup r))$ **YES. All traces start as s0. There r is satisfied. Hence s0 y U r is trivially satisfied. Therefore the formula can rewritten as g U true, which is trivially true.**

(12) $(\neg b \cup b)$ **NO. As in (5) the counterexample is (s0,s1,s3)$^\omega$. Seen as a trace, it never hits b.**

Exercise 02.2

Consider the following statements about CTL formulas:

(1) $(\Box\phi_1) \vee (\Box\phi_2)$ implies $\Box(\phi_1 \vee \phi_2)$

(2) $\Box(\phi_1 \vee \phi_2)$ implies $(\Box\phi_1) \vee (\Box\phi_2)$

For each statement determine if it holds or not.

If the statement does not hold, provide a counterexample (a transition system that satisfies the formula on the left but not the one on the right).

If the statement holds, provide an explanation (similar to the ones in the book or slides).

Solution

(1) $(\Box\phi_1) \vee (\Box\phi_2)$ implies $\Box(\phi_1 \vee \phi_2)$ TRUE

If a trace satisfies the formula on the left, we know that all states satisfy phi1 or they all states satisfy phi2.

Then all states in the trace satisfy at least one of phi1 and phi2 (possibly both).

Hence, the trace satisfies the formula on the right.

Solution

This case is simpler…

(2) $\Box(\phi_1 \vee \phi_2)$ implies $(\Box\phi_1) \vee (\Box\phi_2)$ FALSE

We can easily find a counterexample with two states (the initial state s0 and s1) and two transitions: one from s0 to s1 and one from s1 to s0.

In s0 phi1 holds but not phi2.

In s1 phi2 holds but not phi1.

The only trace of the system satisfies

$$\Box(\phi_1 \vee \phi_2)$$

But none of

$$\Box\phi_1, \Box\phi_2$$

is satisfied

Let p, q be atomic propositions and consider the following formulas

(1) FX p
(2) (F p) ∧ (¬p) ∧ (¬X p) ∧ (¬X X p)
(3) (GF p) ∧ (GF ¬p)
(4) (GF p) ∧ (G¬F p)
(5) (GF p) ∧ (¬GF p)

For each formula… is it satisfiable?

If yes, provide model (a transition system). Make it as small as possible.

If the formula is not satisfiable explain why.

Exercise 02.4: LTL sat appetizer

(1) FX p **SAT (see model in next slide)**

(2) (F p) ∧ (¬p) ∧ (¬X p) ∧ (¬X X p) **SAT (see model in next slide)**

(3) (GF p) ∧ (GF ¬p) **SAT (see model in next slide)**

(4) (GF p) ∧ (G ¬F p) **UNSAT. We observe that Fp and ¬F p are the negation of each other. G phi requires that all states in a trace satisfy phi. A state cannot satisfy a formula phi and its negation. Hence, no such trace exist. Therefore no TS exists that satisfies the formula.**

(5) (GF p) ∧ (¬GF p) **UNSAT. We observe that (GF p) and (¬GF p) are the negation of each other. A trace that satisfies (GF p) does not satisfy (¬GF p) and vice versa. Hence, no trace exists that satisfied the formula, and therefore not TS exists that satisfies the formula.**

See below how the we use black (https://www.black-sat.org/) to check satisfiability of the formulas

```
% black solve --model --formula "F X p "
SAT
Model:
- t = 0: {}
- t = 1: {p}
```

```
% black solve --model --formula "(F p) & (~ p) & (~ X p) & (~ X X p)"
SAT
Model:
- t = 0: {¬p}
- t = 1: {¬p}
- t = 2: {¬p} ⬅ loops here
- t = 3: {p}
```

```
% black solve --model --formula "(G F p) & (G F ~ p)"
SAT
Model:
- t = 0: {}
- t = 1: {p} ⬅ loops here
- t = 2: {¬p}
```

```
% black solve --model --formula "(G F p) & (G ~ F p)"
UNSAT
```

```
% black solve --model --formula "(G F p) & (~ G F p)"
UNSAT
```

Similar results can be obtained with SPOT (https://spot.lre.epita.fr/app/)