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.

$$\Downarrow$$

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

more detailed

$I \models S$

more abstract

"implementation"
(system model)

"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

P1		P2
loop	||	loop
out: x1 := 1; last := 1		out: x2 := 1; last := 2
req: await x2 = 0 or last = 2		req: await x1 = 0 or last = 1
in: x1 := 0		in: x2 := 0
end loop.		end loop.

Model-checking problem

$$I \models S$$

system model

satisfaction relation

system property

Model-checking problem

$$I \models S$$

system model

satisfaction relation

system property

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

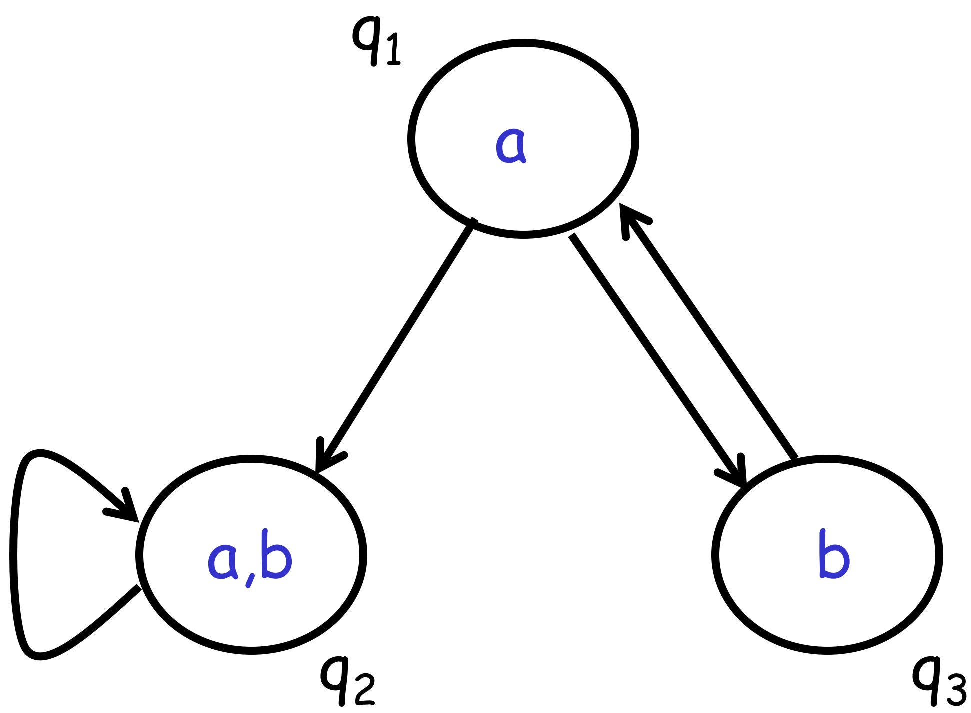
I/O automata

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.


q1
a
a,b
q2
b
q3

State-transition graph

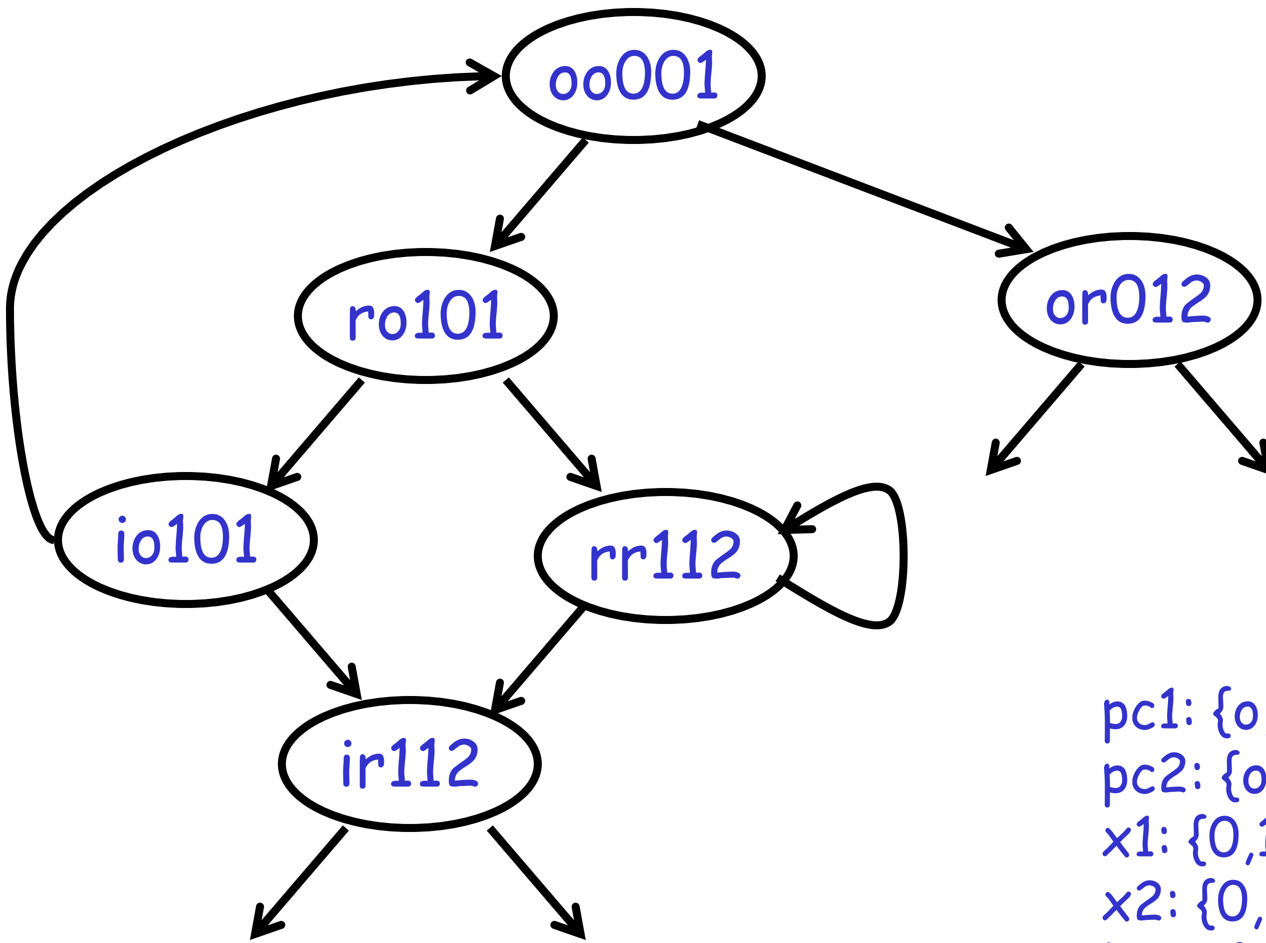
Q	set of states	$\{q_1,q_2,q_3\}$
A	set of atomic observations	$\{a,b\}$
$\rightarrow \subseteq Q \times Q$	transition relation	$q_1 \rightarrow q_2$
$[\ ]: Q \rightarrow 2^A$	observation function	$[q_1] = \{a\}$

set of observations

Mutual-exclusion protocol

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

          P1                                        P2
```



pc1: {o,r,i}
pc2: {o,r,i}
x1: {0,1}
x2: {0,1}
last: {1,2}

$3 \cdot 3 \cdot 2 \cdot 2 \cdot 2 = 72$ states

The translation from a system description to a state-transition graph usually involves an exponential blow-up !!!

e.g., n boolean variables $\Rightarrow$ 2^n 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")

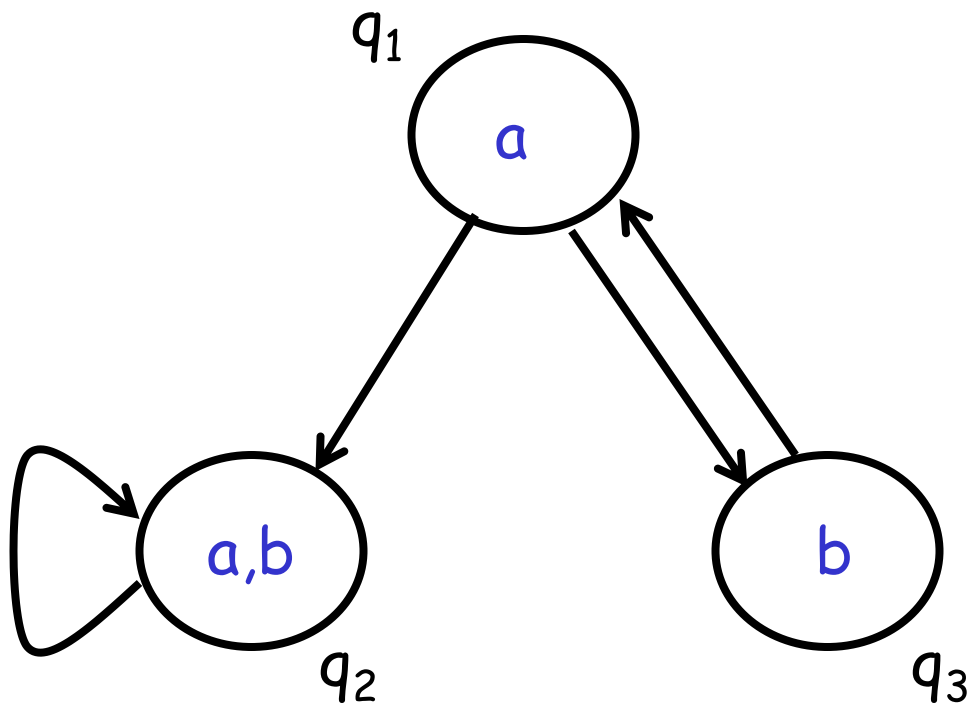
Safety vs. liveness for state-transition graphs

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: $q_1 \to q_3 \to q_1 \to q_3 \to q_1 \to q_2 \to q_2 \to$

Trace: $a \to b \to a \to b \to a \to a,b \to a,b \to$

State-transition graph $S = (Q, A, \to, [\,])$

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

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

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

Infinite traces: $infTraces(S) \subseteq (2^A)^\omega$

Safety: the properties that can be checked on finRuns

Liveness: the properties that cannot be checked on finRuns

This is much easier.

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)

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

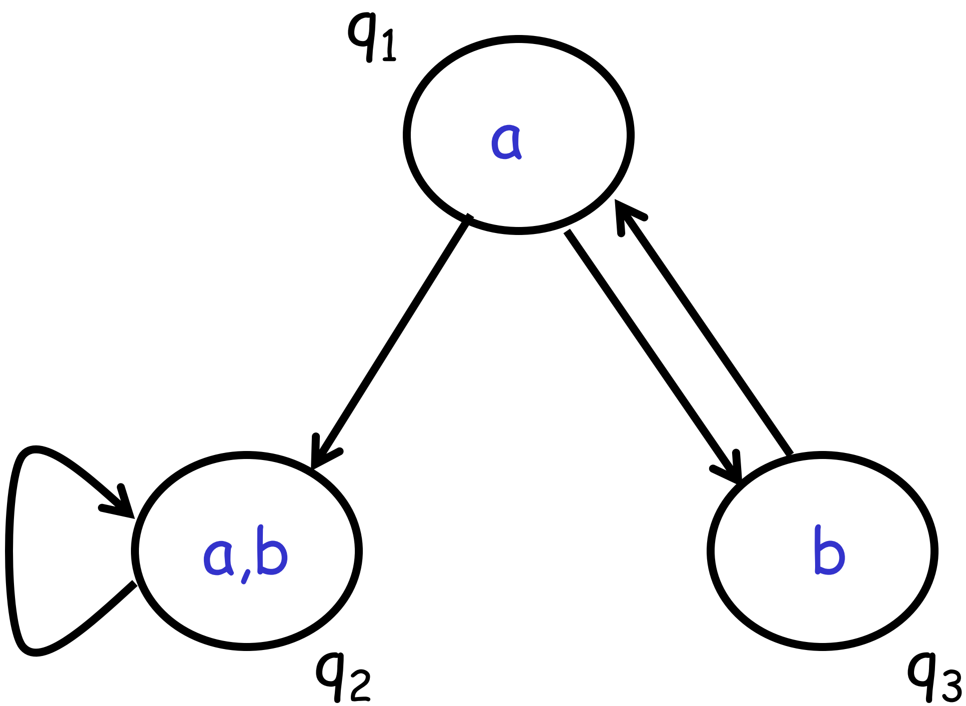
Example: Starvation freedom

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

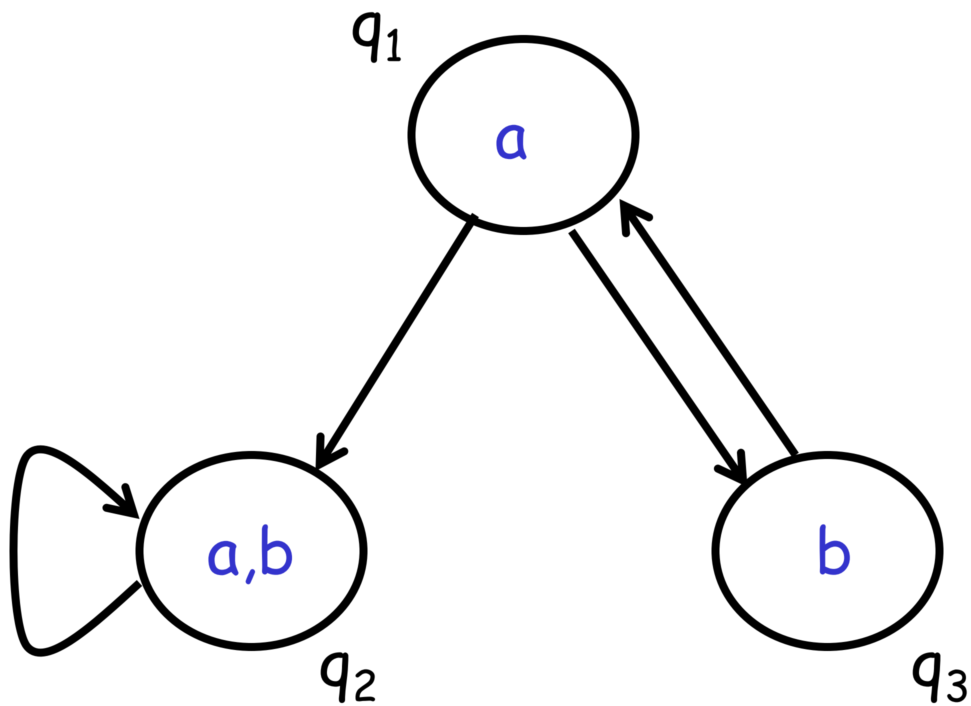
Example: Starvation freedom

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

$\Leftarrow^*$
closure

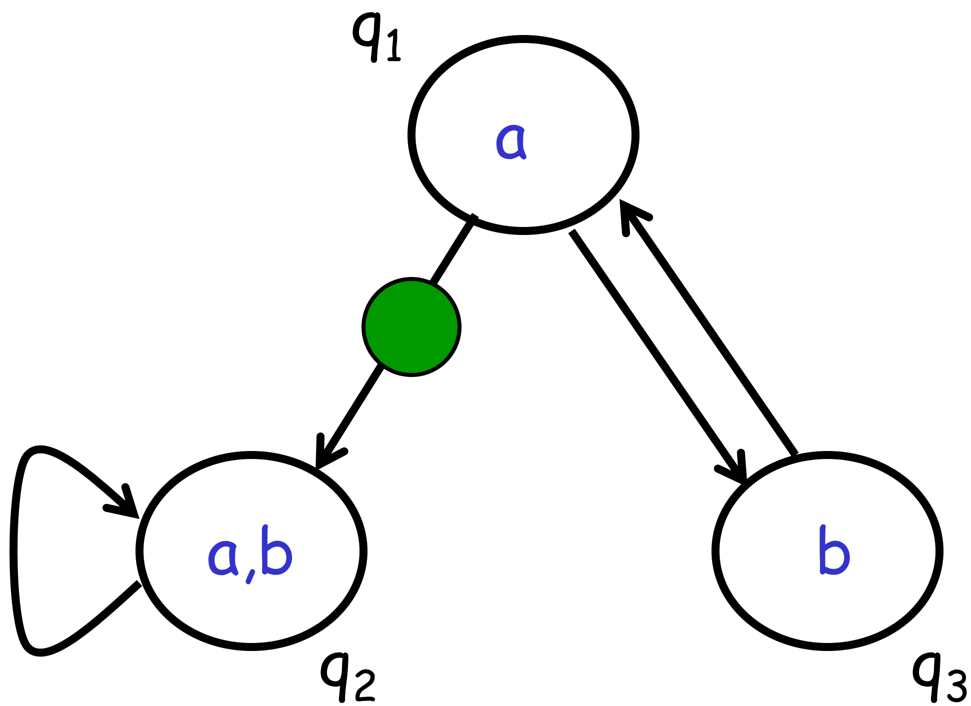
*finite branching

For state-transition graphs,
all properties are safety properties !

Example: Starvation freedom

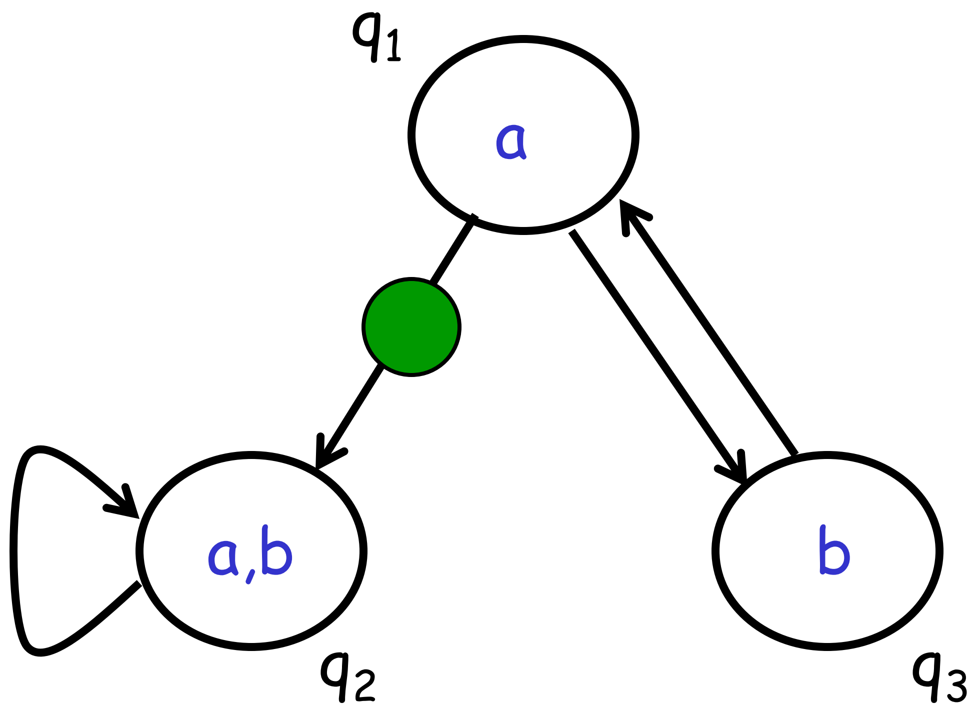
Whenever process P1 wants to enter the critical section, provided process P2 never stays in the critical section forever, P1 gets to enter eventually.

Liveness



Fairness constraint:

the green transition cannot be ignored forever



Without fairness: infRuns = $q_1 \, (q_3 \, q_1)^* \, q_2^\omega \cup (q_1 \, q_3)^\omega$

With fairness: infRuns = $q_1 \, (q_3 \, q_1)^* \, q_2^\omega$

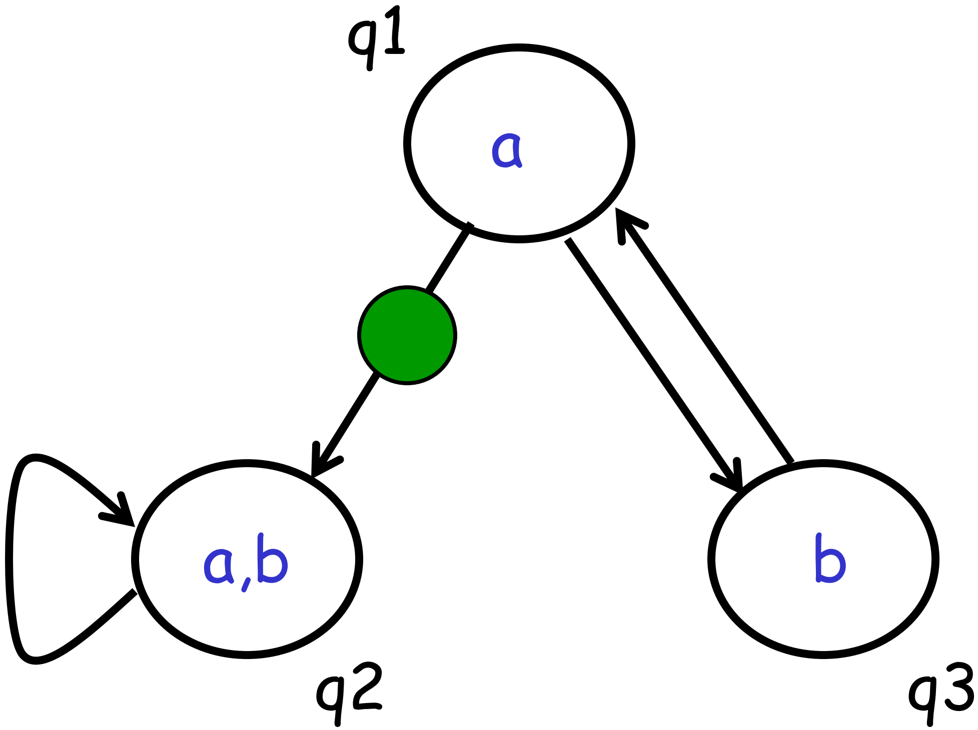
Two important types of fairness

1 Weak (Buchi) fairness:

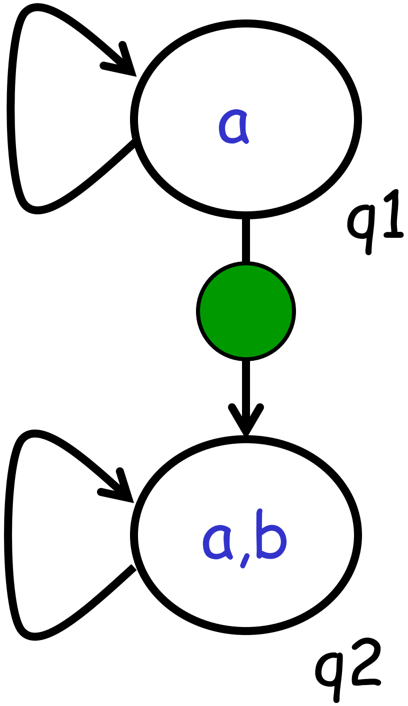
a specified set of transitions cannot be enabled forever without being taken

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 $S = (Q, A, \rightarrow, [], WF, SF)$

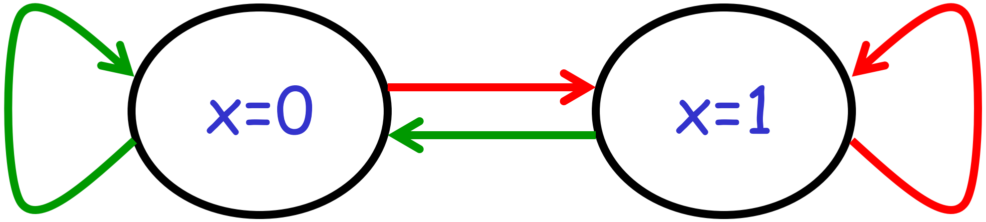
WF set of weakly fair actions

SF set of strongly fair actions

where each action is a subset of $\rightarrow$

Weak fairness comes from modeling concurrency

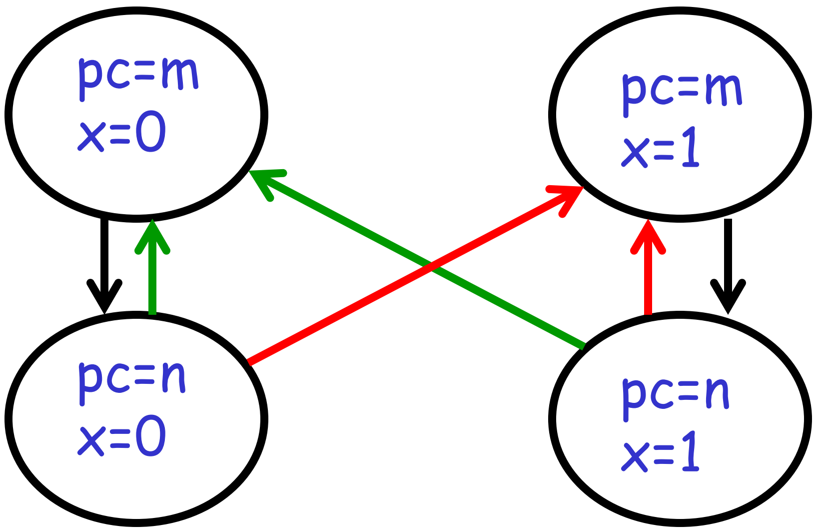
loop x:=0 end loop. || loop x:=1 end loop.



Weakly fair action
Weakly fair action

Strong fairness comes from modeling choice

```
loop    m:
        n:   x:=0 | x:=1
end loop.
```



Strongly fair action
Strongly fair action

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.

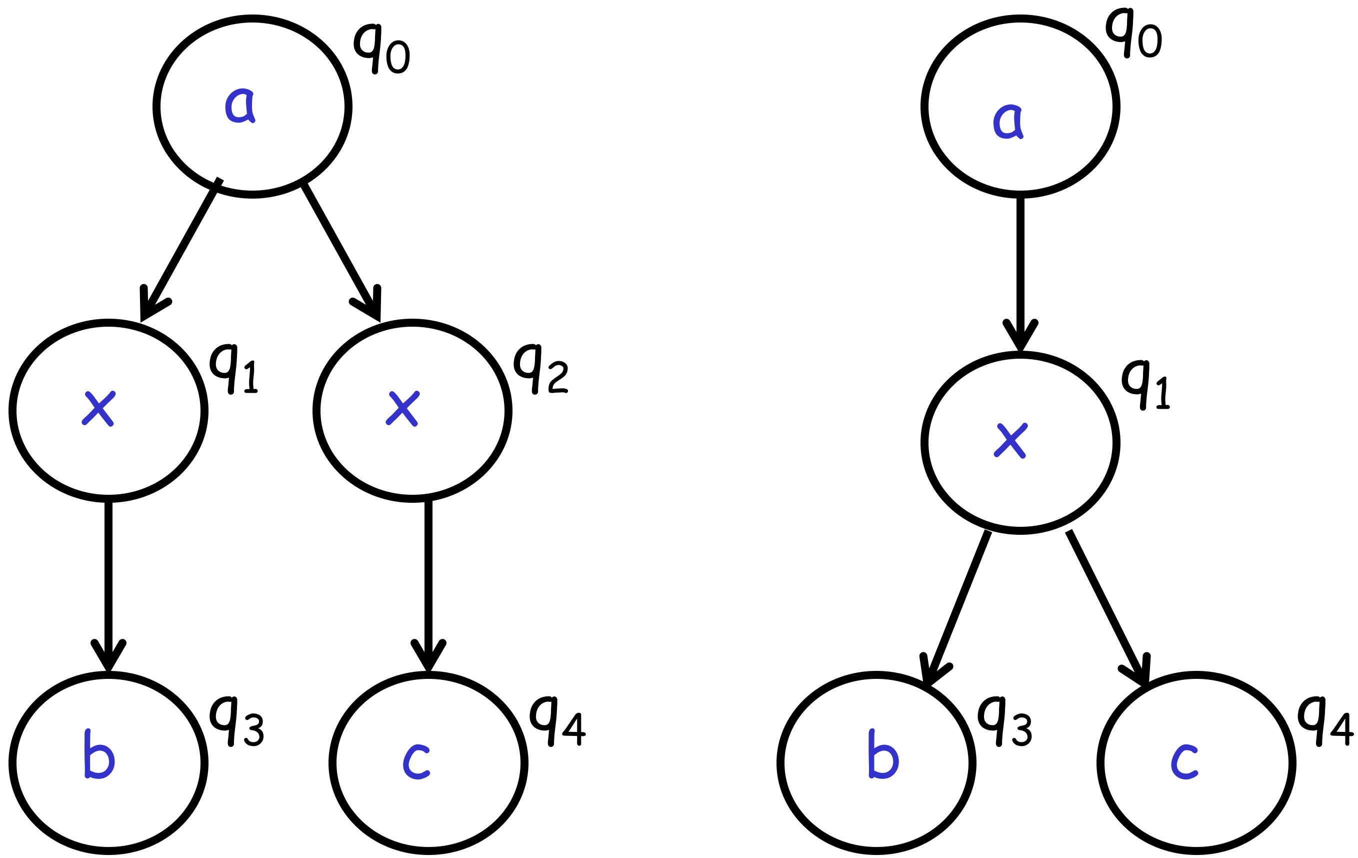
Fair state-transition graph $S = (Q, A, \to, [], WF, SF)$

Finite runs:	$finRuns(S) \subseteq Q^*$
Infinite runs:	$infRuns(S) \subseteq Q^\omega$
Finite traces:	$finTraces(S) \subseteq (2^A)^*$
Infinite traces:	$infTraces(S) \subseteq (2^A)^\omega$

Branching vs. linear time

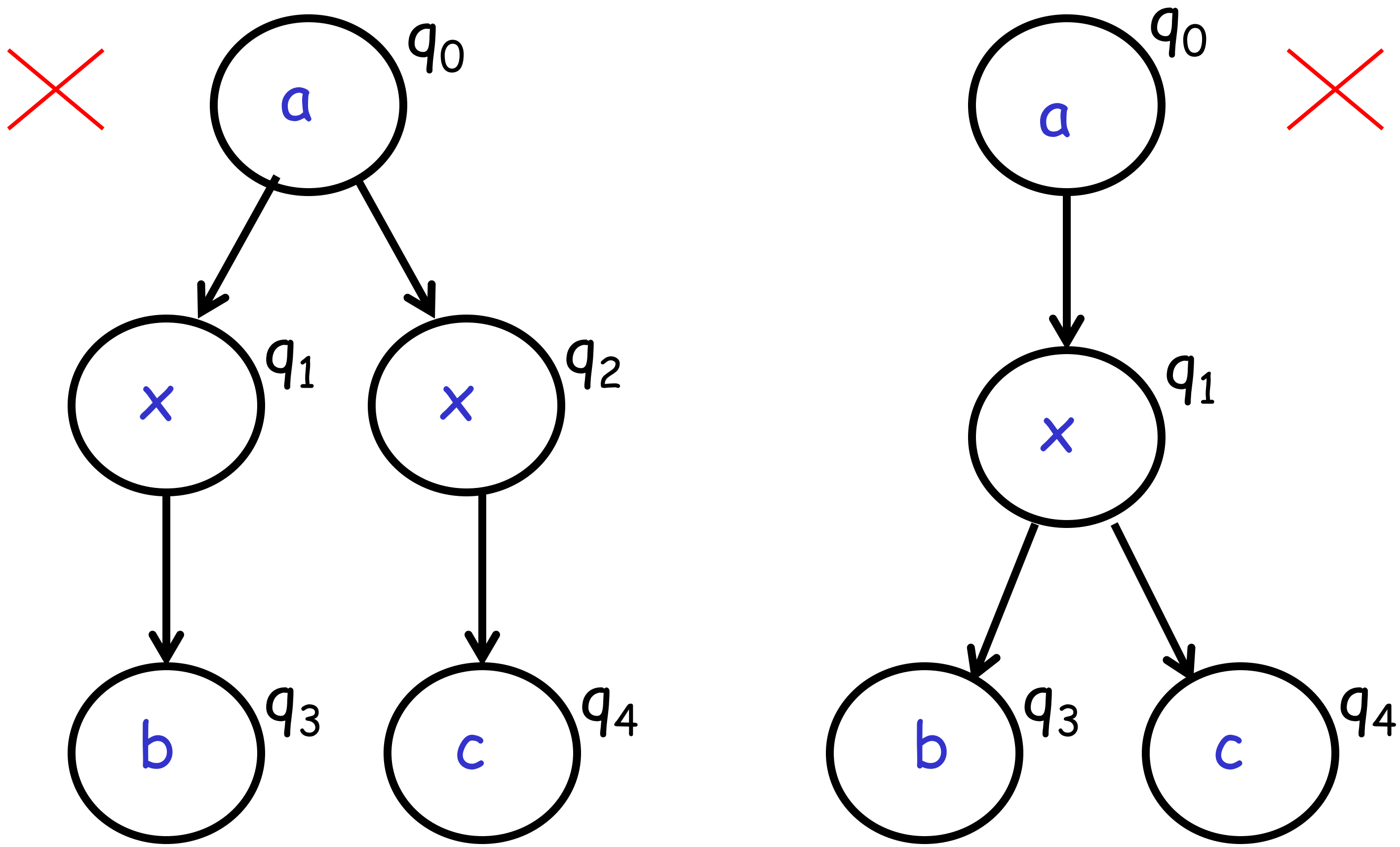
Linear time: the properties that can be checked on infTraces

Branching time: the properties that cannot be checked on infTraces

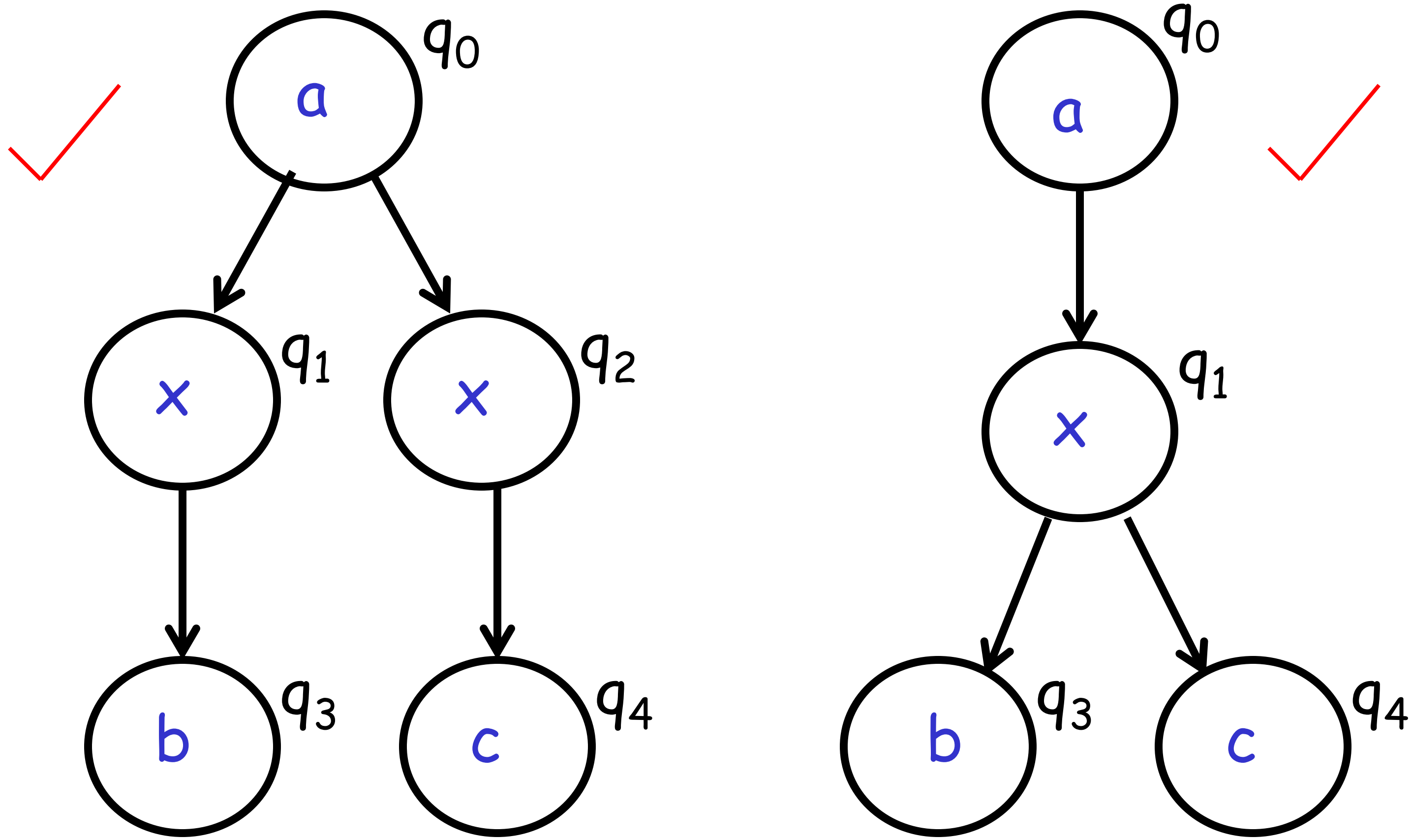


Same traces $\{axb, axc\}$

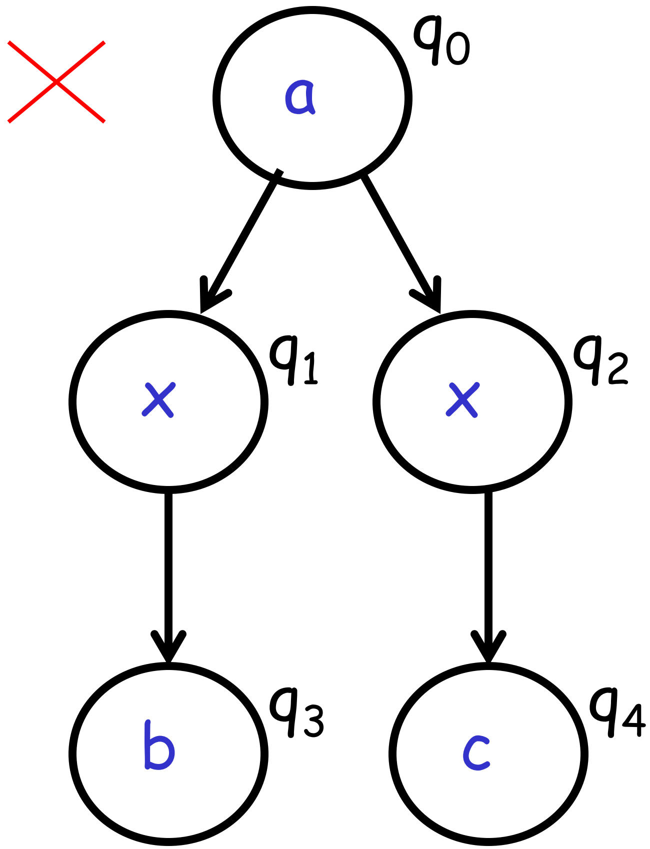
Different runs $\{q_0\, q_1\, q_3,\ q_0\, q_2\, q_4\}$, $\{q_0\, q_1\, q_3,\ q_0\, q_1\, q_4\}$

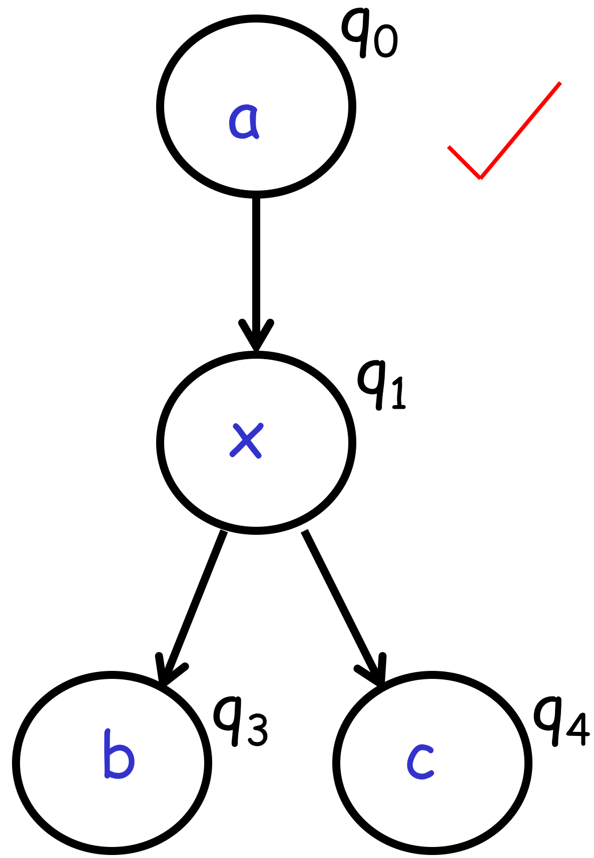


Linear-time:
In all traces, an x must happen immediately followed by b

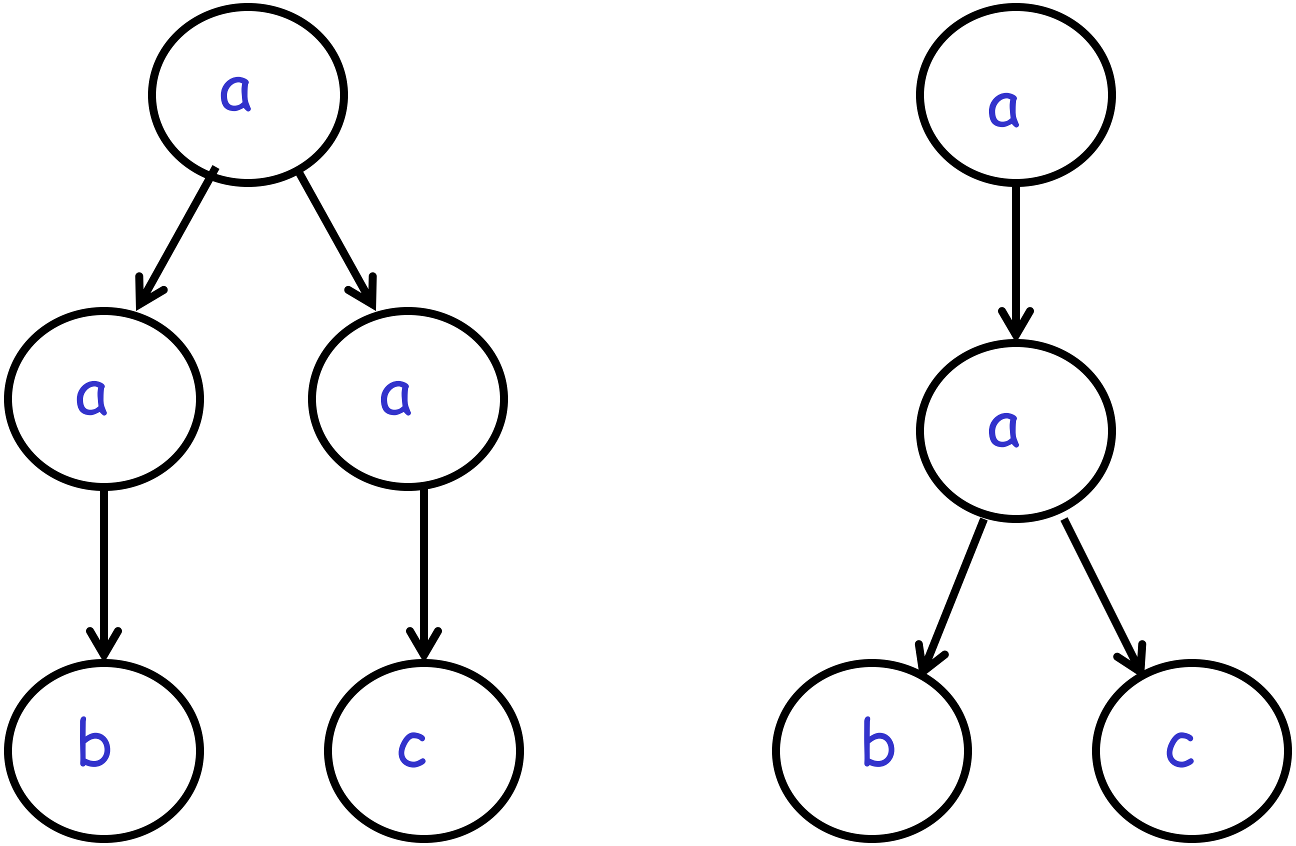


Linear-time:
In all traces, an x must happen immediately followed by b or c





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

	Linear	Branching
Safety		STL
Liveness	LTL	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 φ ? $M \models \varphi$

Propositional logics:

1. boolean variables (a,b) & boolean operators ($\wedge$,$\neg$)
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

atomic observations

Propositional logics:

1. boolean variables (a,b) & boolean operators ($\wedge$,$\neg$)
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

observations

state-transition graph ("Kripke structure")

STL Syntax

$$\varphi \;::=\; a \;\mid\; \varphi \wedge \varphi \;\mid\; \neg\,\varphi \;\mid\; \exists\bigcirc\,\varphi \;\mid\; \varphi\,\exists U\,\varphi$$

boolean operators

boolean variable
(atomic observation)

modal operators

STL Model

(K, q)

state-transition graph
(Kripke structure)

state of K

STL Semantics

$(K,q) \models a$ iff $a \in [q]$

$(K,q) \models \varphi \wedge \psi$ iff $(K,q) \models \varphi$ and $(K,q) \models \psi$

$(K,q) \models \neg\varphi$ iff not $(K,q) \models \varphi$

$(K,q) \models \exists\bigcirc\varphi$ iff exists q' s.t. $q \rightarrow q'$ and $(K,q') \models \varphi$

$(K,q) \models \varphi\, \exists U\, \psi$ iff exists $q = q0 \rightarrow q1 \rightarrow \ldots \rightarrow qn$.

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

Defined modalities

$\exists\bigcirc$	EX	exists next
$\forall\bigcirc\,\varphi = \neg\exists\bigcirc\neg\varphi$	AX	forall next
$\exists U$	EU	exists until
$\exists\Diamond\,\varphi = \text{true}\,\exists U\,\varphi$	EF	exists eventually
$\forall\Box\,\varphi = \neg\,\exists\Diamond\,\neg\varphi$	AG	forall always

$$\varphi\forall W\psi = \neg\,(\,(\neg\psi)\,\exists U\,(\neg\varphi \wedge \neg\psi))$$

	AW	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. $q = q_0 \to q_1 \to q_2 \to \ldots,$
   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 ???

$(K,q) \models \varphi \forall W \psi$

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

$(K,q) \models \varphi \forall W \psi$ iff

$(K,q) \models \neg ((\neg\psi) \exists U (\neg\varphi \wedge \neg\psi))$ iff

$\neg$ (exists $q = q_0 \to q_1 \to \ldots \to q_n$.
for all $0 \leq i < n$. $(K,q_i) \models \neg \psi$ and $(K,q_n) \models \neg\varphi \wedge \neg\psi$) iff

for all $q = q_0 \to q_1 \to \ldots \to q_n$.
exists $0 \leq i < n$. $(K,q_i) \models \psi$ or $(K,q_n) \models \varphi \vee \psi$ iff

for all $q = q_0 \to q_1 \to \ldots \to q_n$.
exists $0 \leq i \leq n$. $(K,q_i) \models \psi$ or $(K,q_n) \models \varphi$ iff

for all $q = q_0 \to q_1 \to \ldots \to q_n$.
$(K,q_n) \models \neg\varphi \Rightarrow$ exists $0 \leq i \leq n$. $(K,q_i) \models \psi$

Important safety properties

Invariance $\forall \Box\, a$

Sequencing $a \;\forall W\; b \;\forall W\; c \;\forall W\; d$

$= \; a \;\forall W\; (b \;\forall W\; (c \;\forall W\; d))$

Important safety properties: mutex protocol

Invariance $\forall\Box\,\neg\,(pc1{=}in \wedge pc2{=}in)$

Sequencing $\forall\Box\,(\,pc1{=}req \;\Rightarrow$

$(pc2{\neq}in)\ \forall W\ (pc2{=}in)\ \forall W\ (pc2{\neq}in)\ \ \forall W\ (pc1{=}in))$

Branching properties

Deadlock freedom $\forall\Box\,\exists\bigcirc$ true

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

$\forall\Box\,(pc1{=}req \Rightarrow \exists\Diamond\,(pc1{=}in))$

CTL (Computation Tree Logic)

- -safety & liveness
- -branching time

[Clarke & Emerson; Queille & Sifakis 1981]

CTL Syntax

$$\varphi \; ::= \; a \;\mid\; \varphi \wedge \varphi \;\mid\; \neg \varphi \;\mid\; \exists\bigcirc \varphi \;\mid\; \varphi\, \exists\mathsf{U}\, \varphi \;\mid\; \exists\square\varphi$$

CTL Model

(K, q)

fair state-transition graph

state of K

CTL Semantics

$(K,q) \models \exists\Box\,\varphi$ iff exist $q_0, q_1, \ldots$ 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

$\exists\Box$	EG	exists always
$\forall\Diamond\,\varphi = \neg\exists\Box\neg\varphi$	AF	forall eventually

$$\varphi\,\exists W\,\psi = (\varphi\,\exists U\,\psi) \vee (\exists\Box\,\varphi)$$

$$\varphi\,\forall U\,\psi = (\varphi\,\forall W\,\psi) \wedge (\forall\Diamond\psi)$$

Important liveness property

Response $\forall\Box\,(a \Rightarrow \forall\Diamond\, b)$

$\forall\Box\,(pc1{=}req \Rightarrow \forall\Diamond\,(pc1{=}in))$