Computer Aided Verification
計算機輔助驗證

Model Checking (Part I)
模型檢驗 (一)

Pao-Ann Hsiung
Department of Computer Science and Information Engineering
National Chung Cheng University, Taiwan
熊博安
國立中正大學 資訊工程研究所

These slide contents are adapted from the slides of Professor Thomas Henzinger.

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 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

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

```
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

Model-checking problem

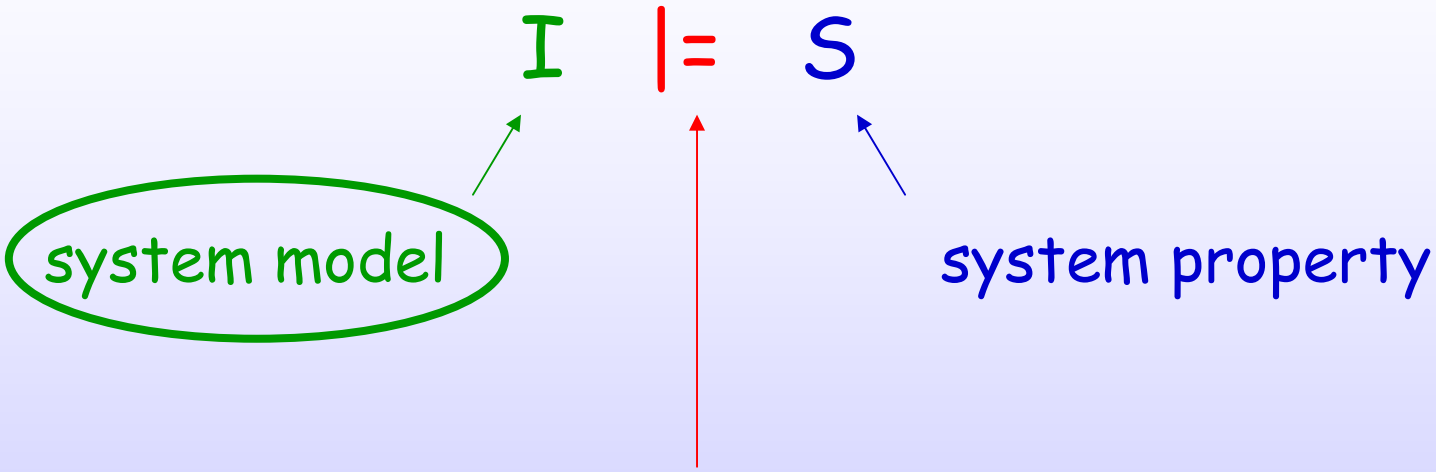
$$I \models S$$

system model

satisfaction relation

system property

Model-checking problem



Important decisions when choosing a system model

-variable-based vs. event-based

-interleaving vs. true concurrency

-synchronous vs. asynchronous interaction

-clocked vs. speed-independent progress

-etc.

Particular combinations of choices yield

CSP

Petri nets

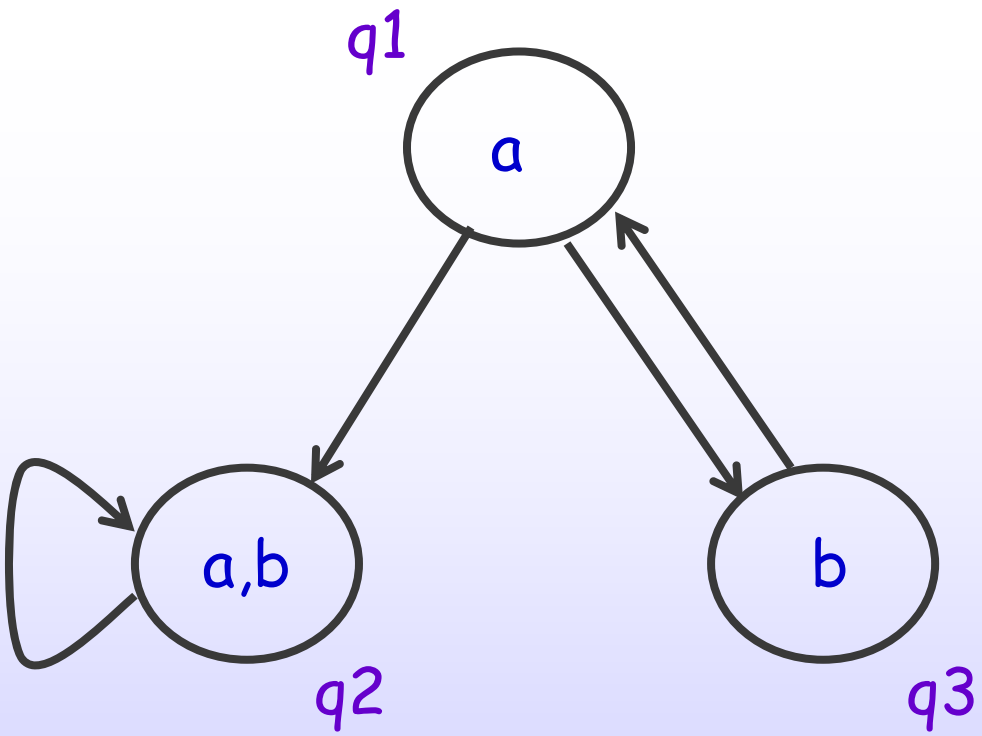
I/O automata

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
b
q2
q3

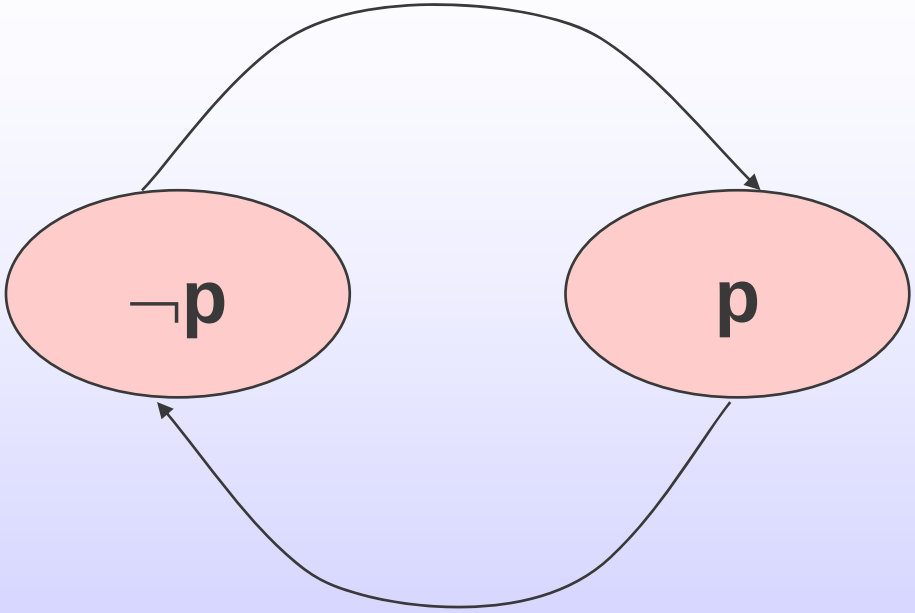
State-Transition Graphs Kripke Structures (KS)

Q	set of states	$\{q1,q2,q3\}$
A	set of observations	$\{a,b\}$
$\rightarrow \subseteq Q \times Q$	transition relation	$q1 \rightarrow q2$
$[\ ]: Q \rightarrow 2^A$	observation function	$[q1] = \{a\}$

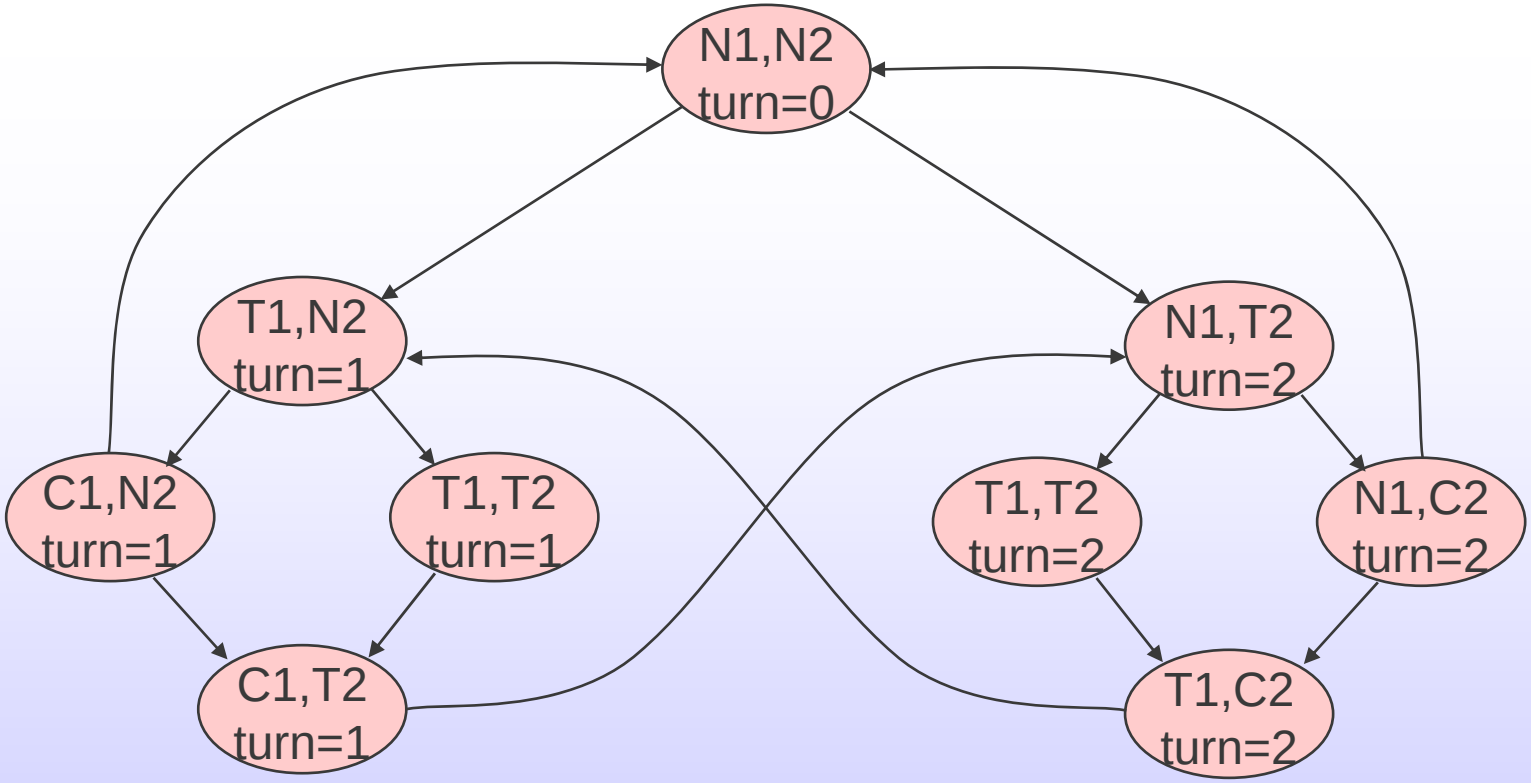
K = (Q, A, →, [ ])

Kripke Structure of Programs

```
repeat
      p := true;
      p := false;
end
```



Mutual Exclusion KS



N = noncritical, T = trying, C = critical

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."

State-transition graphs are not necessarily finite-state, but they don't handle well:

-recursion (need push-down models)

-environment interaction (need game models)

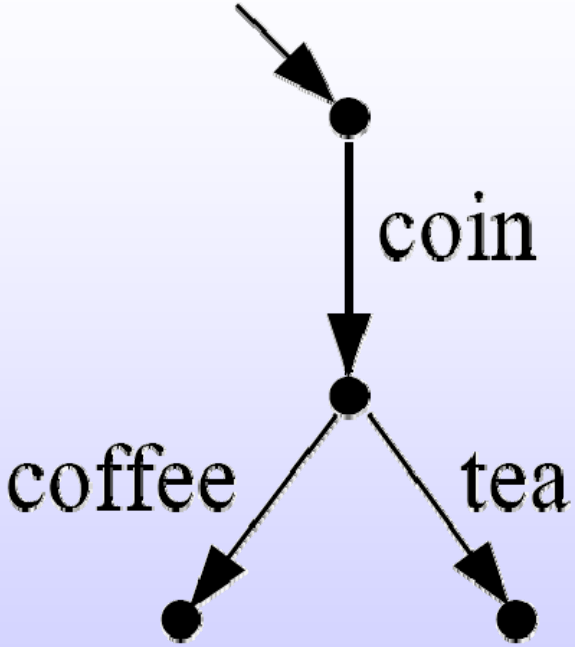
-process creation

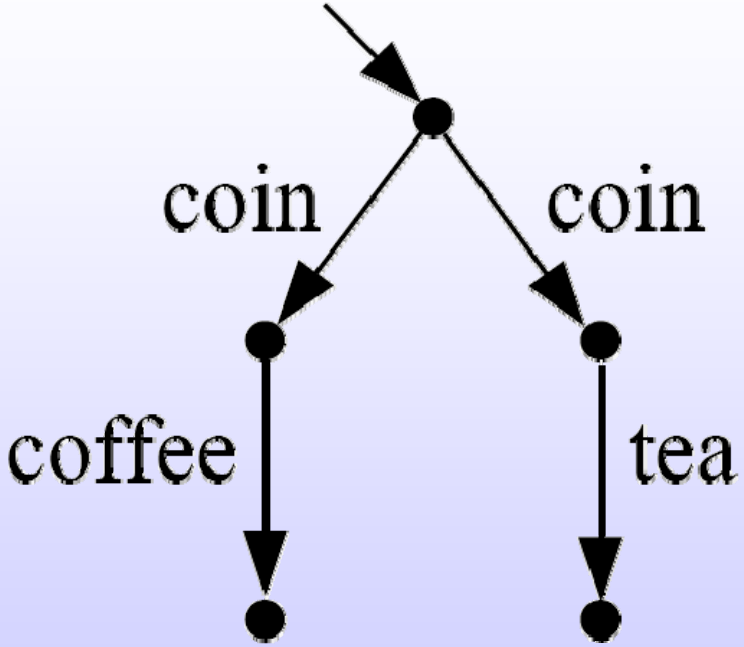
Labeled Transition Systems (LTS)

Q	set of states	$\{q1,q2,q3\}$
Act	set of actions	$\{a,b\}$
$\rightarrow \subseteq Q \times Act \times Q$	transition relation	$q1 \rightarrow q2$

L = (Q, Act, →)

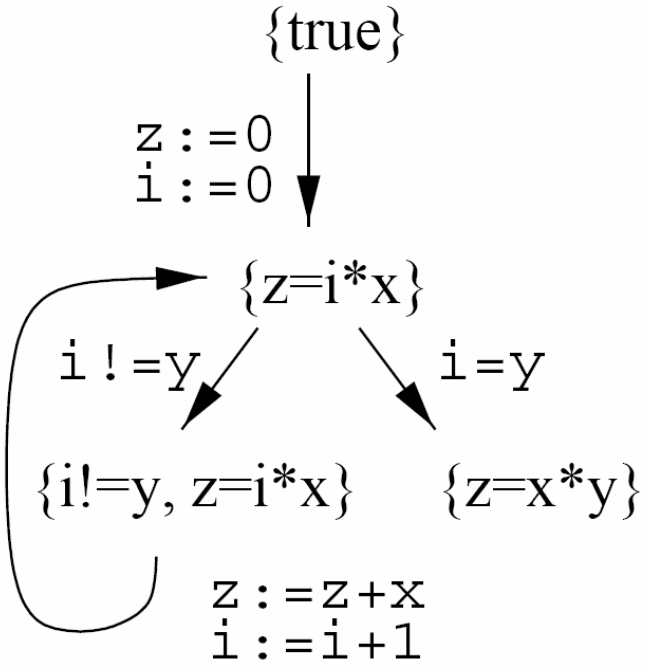
Vending Machine LTS

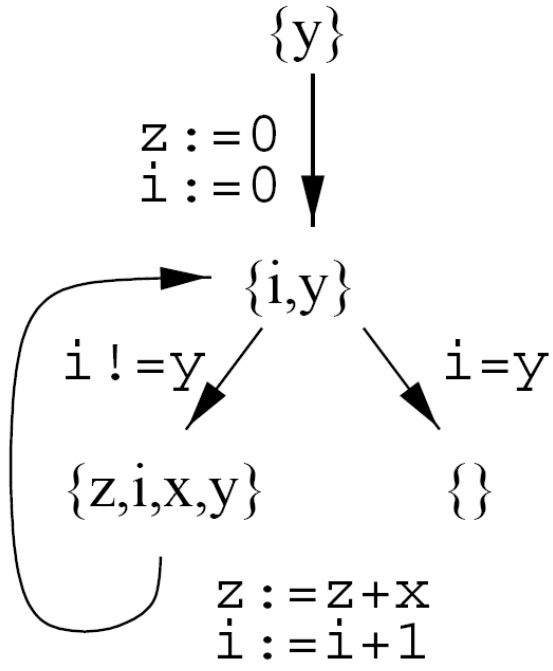




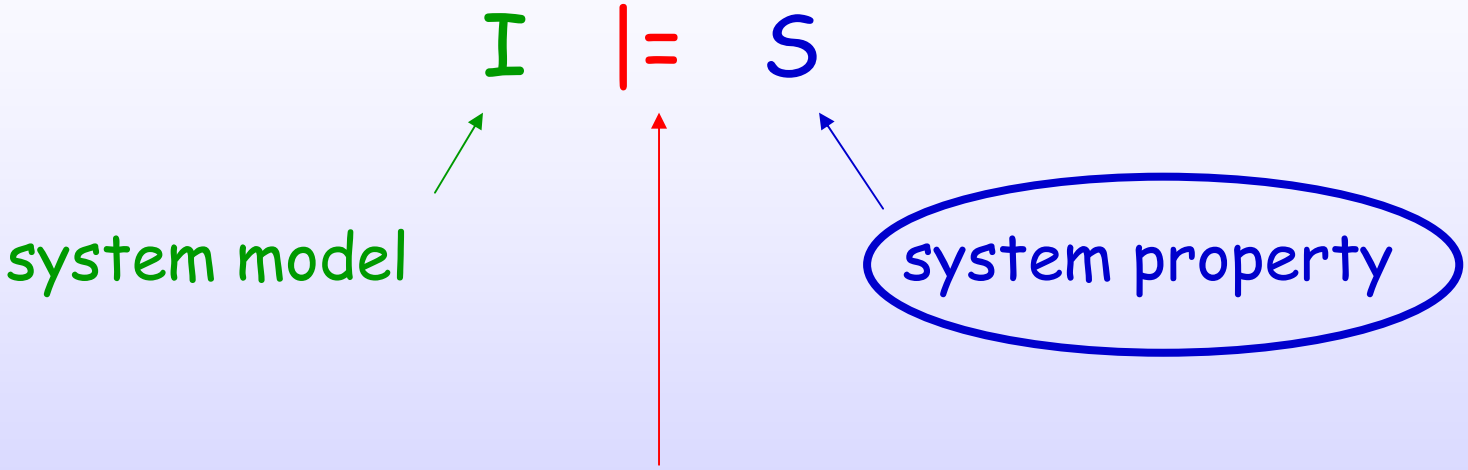
Kripke Transition Systems

KTS = KS + LTS





Model-checking problem



Three important decisions when choosing system properties

1. operational vs. declarative:
   automata vs. logic
2. may vs. must:
   branching vs. linear time
3. prohibiting bad vs. desiring good behavior:
   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

induction on control flow

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

Liveness: the program will produce a result ("termination")

well-founded induction on data

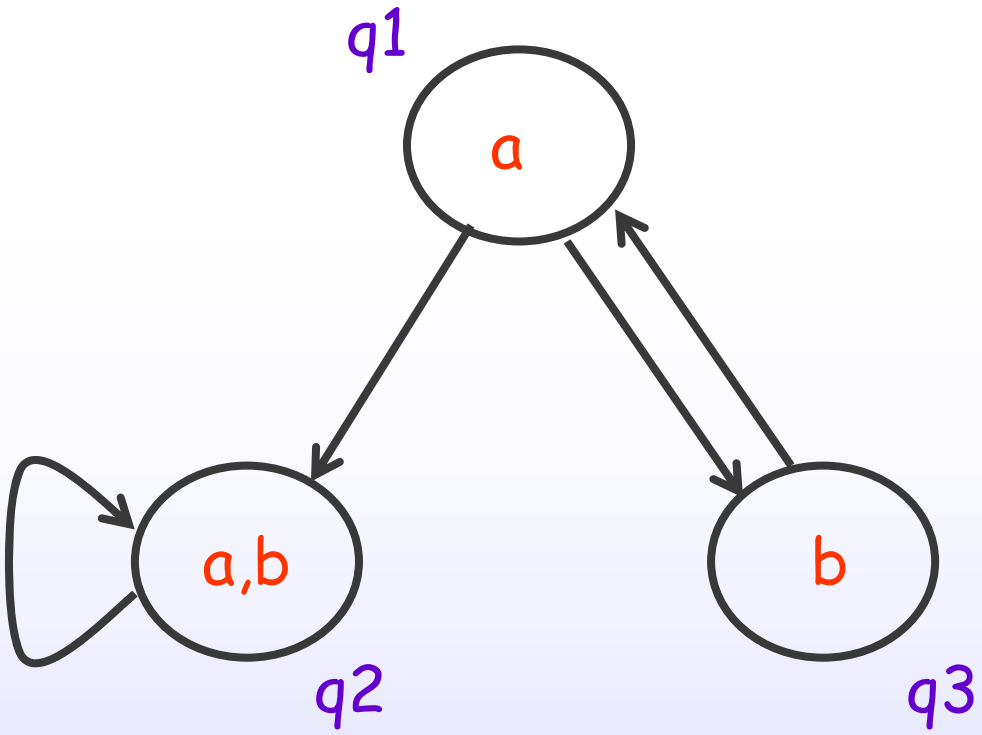
Safety vs. liveness for state-transition graphs

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: $q1 \rightarrow q3 \rightarrow q1 \rightarrow q3 \rightarrow q1 \rightarrow q2 \rightarrow q2 \rightarrow$

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

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

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

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

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.

Safety

Example: Bounded overtaking

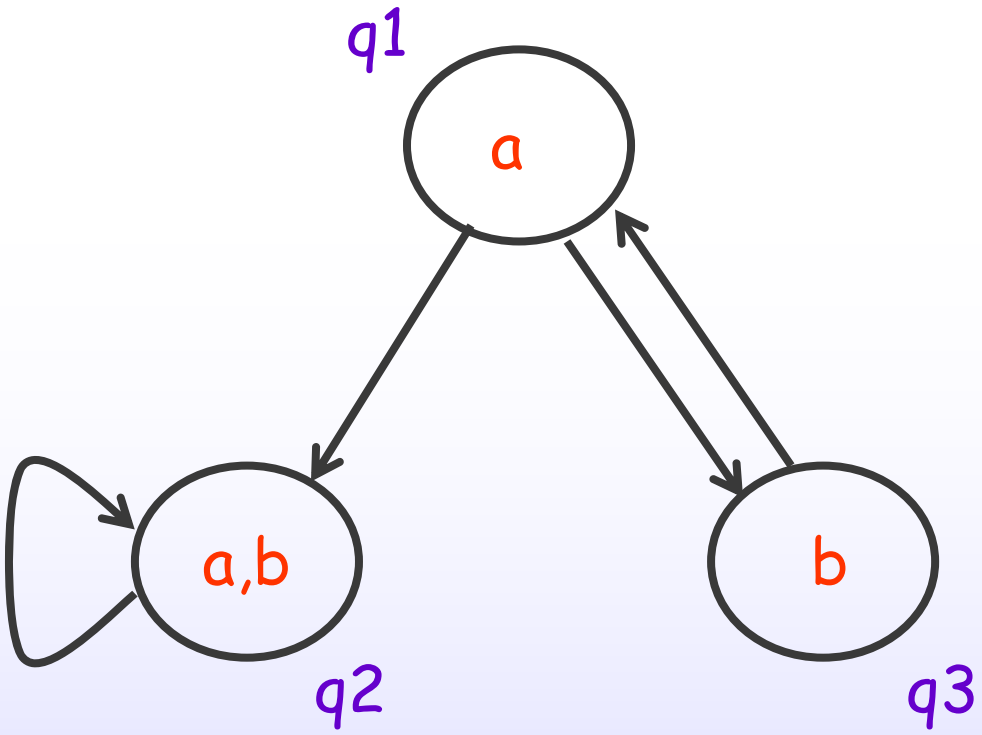
Whenever process P1 wants to enter the critical section, then process P2 gets to enter at most once before process P1 gets to enter.

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.

Liveness



infRuns $\Rightarrow$ finRuns

$\Leftarrow$

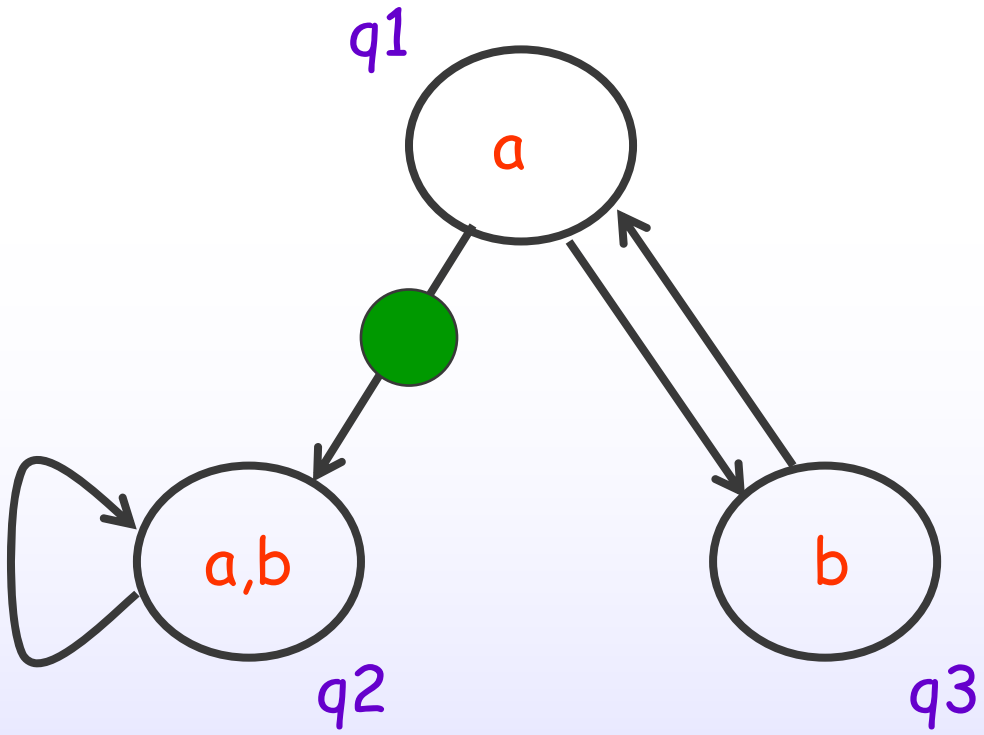
closure

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

Example: Starvation freedom

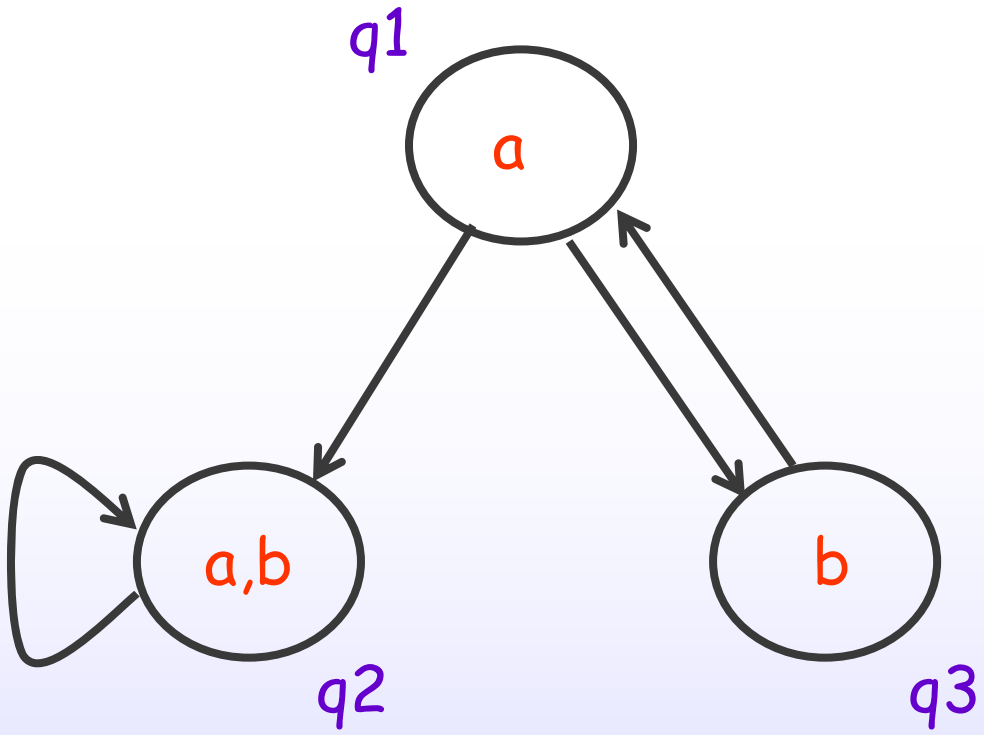
Whenever process P1 wants to enter the critical section, provided process P2 never stays in the critical section forever, P1 gets to enter eventually.

Liveness



Fairness constraint:

the green transition cannot be ignored forever



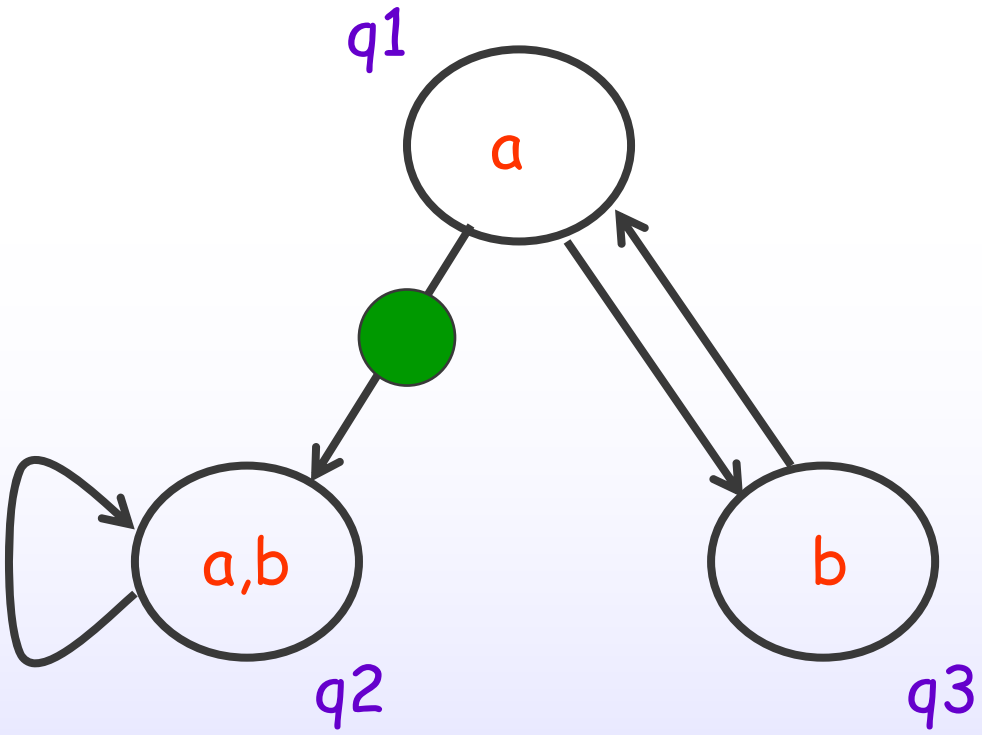
Without fairness: infRuns = $q1\ (q3\ q1)^*\ (q2)^\omega \cup (q1\ q3)^\omega$

With fairness: infRuns = $q1\ (q3\ q1)^*\ (q2)^\omega$

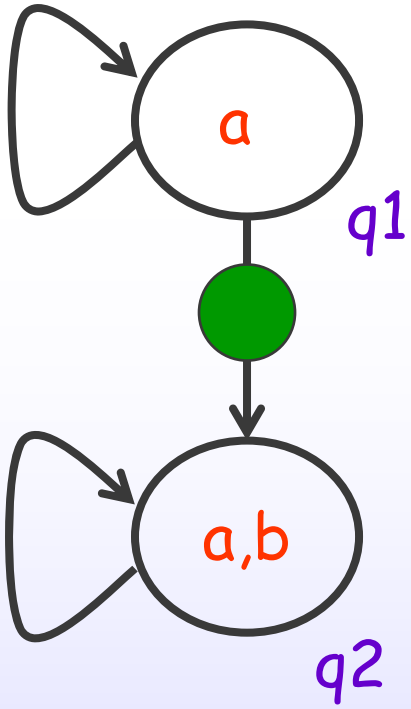
Two important types of fairness

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

Weak fairness is sufficient for asynchronous models ("no process waits forever if it can move").

Strong fairness is necessary for modeling synchronous interaction (rendezvous).

Strong fairness makes model checking more difficult.

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 $\rightarrow$

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. operational vs. declarative:
   automata vs. logic
2. may vs. must:
   branching vs. linear time
3. prohibiting bad vs. desiring good behavior:
   safety vs. liveness

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

Branching vs. linear time

Branching time: something may (or may not) happen
(e.g., every req may be followed by grant)

Linear time: something must (or must not) happen
(e.g., every req must be followed by grant)

One is rarely interested in may properties,

but certain may properties are easy to model check, and they imply interesting must properties.

(This is because unlike must properties, which refer only to observations, may properties can refer to states.)

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

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

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

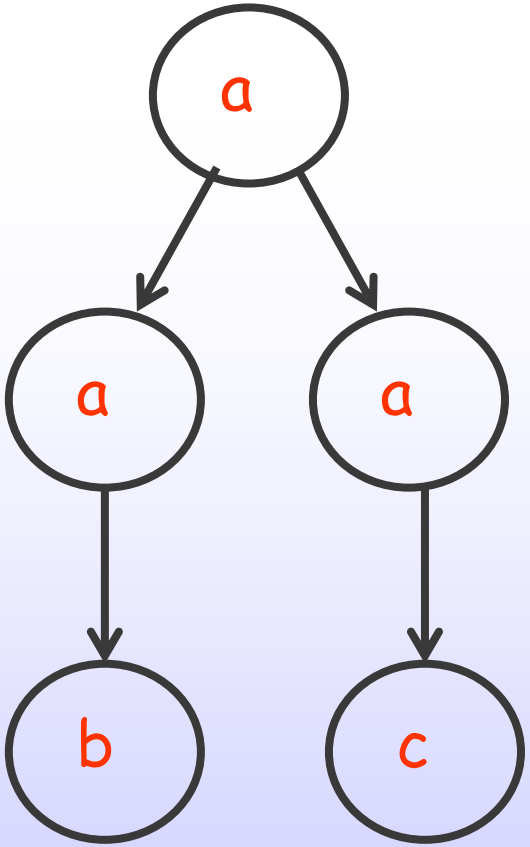
Finite traces: $finTraces(S) \subseteq (2^A)^*$

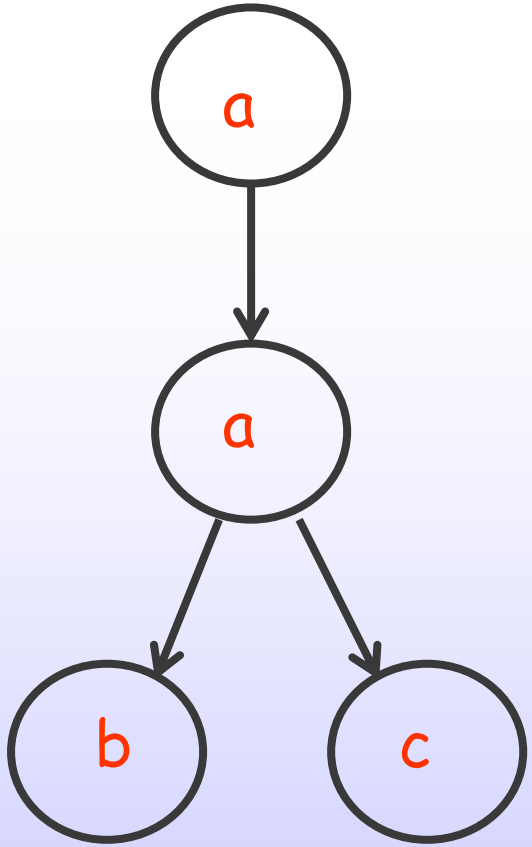
Infinite traces: $infTraces(S) \subseteq (2^A)^\omega$

Linear time: the properties that can be checked on infTraces

Branching time: the properties that cannot be checked on infTraces

	Linear	Branching
Safety	finTraces	finRuns
Liveness	infTraces	infRuns





Same traces, different runs

Observation a may occur.

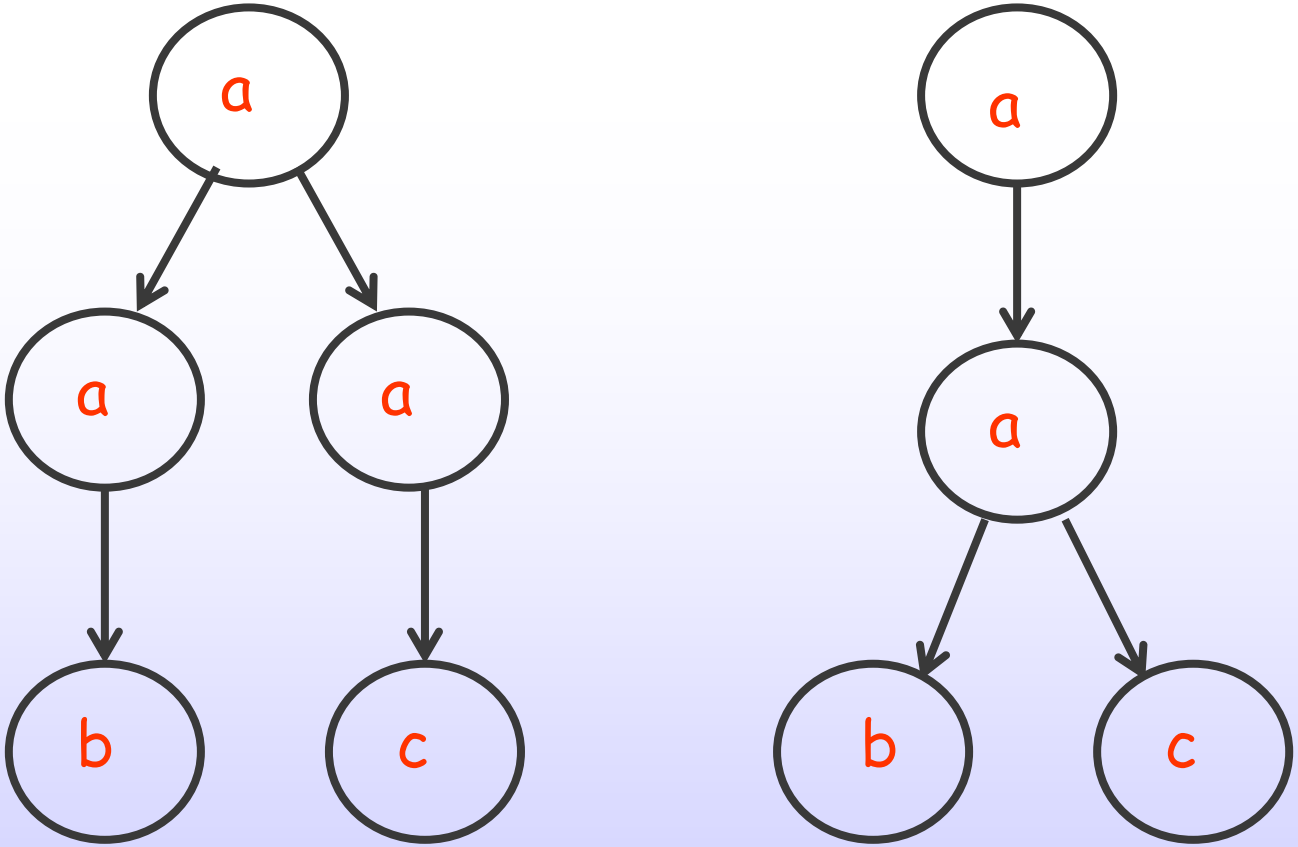
||

It is not the case that a must not occur.

Linear

We may reach an a from which we must not reach a b .

Branching



Same traces, different runs (different trace trees)

Linear time is conceptually simpler than branching time (words vs. trees).

Branching time is often computationally more efficient.

(Because branching-time algorithms can work with given states,
whereas linear-time algorithms often need to "guess" sets of possible states.)

Three important decisions when choosing system properties

1. operational vs. declarative:
   automata vs. logic
2. may vs. must:
   branching vs. linear time
3. prohibiting bad vs. desiring good behavior:
   safety vs. liveness

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

Logics

	Linear	Branching
Safety		SafeTL
Liveness	LTL	CTL

Automata

Safety:	finite automata
Liveness:	omega automata
Linear:	language containment
Branching:	simulation

Automata

Safety: finite automata

Liveness: omega automata

Linear: language containment for word automata

Branching: language containment for tree automata

System property: 2x2x2 choices

-safety (finite runs) vs. liveness (infinite runs)

-linear time (traces) vs. branching time (runs)

-logic (declarative) vs. automata (executable)

Defining a logic

1. Syntax:

   What are the formulas?

2. Semantics:

   What are the models?

   Does model M satisfy formula φ ? M |= φ

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")

CTL (Computation Tree Logic)

-safety & liveness

-branching time

-logic

[Clarke & Emerson; Queille & Sifakis 1981]

CTL Syntax

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

boolean operators

boolean variable
(atomic observation)

modal operators

CTL Model

(K, q)

fair state-transition graph

state of K

CTL 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 exist $q_0, \ldots, q_n$ s.t.
1. $q = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n$
2. for all $0 \leq i < n$, $(K,q_i) \models \varphi$
3. $(K,q_n) \models \psi$

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 (safety)

$\exists\bigcirc$	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)

Defined modalities (liveness)

$\exists\Box$	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 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 $\forall\Box\ \neg\ (in\_cs1 \wedge in\_cs2)$

Sequencing $\forall\Box\ (\ req\_cs1\ \Rightarrow$

$\neg in\_cs2\ \forall W\ in\_cs2\ \forall W\ \neg in\_cs2\ \forall W\ in\_cs1\ )$

Branching properties

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

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

$\forall\Box\,(req\_cs1 \Rightarrow \exists\Diamond\, in\_cs1)$

Important liveness property

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

$\forall\Box\,(req\_cs1 \Rightarrow \forall\Diamond\, in\_cs1)$

If only universal properties are of interest,

why not omit the path quantifiers?

LTL (Linear Temporal Logic)

-safety & liveness

-linear time

-logic

[Pnueli 1977; Lichtenstein & Pnueli 1982]

LTL Syntax

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

LTL Model

infinite trace $t = t_0\ t_1\ t_2\ \ldots$

Language of deadlock-free state-transition graph K at state q :

L(K,q) ... set of infinite traces of K starting at q

$(K,q) \models \forall\varphi$ iff for all $t \in L(K,q)$, $t \models \varphi$

$(K,q) \models \exists\varphi$ iff exists $t \in L(K,q)$, $t \models \varphi$

LTL Semantics

$t \models a$ iff $a \in t_0$

$t \models \varphi \wedge \psi$ iff $t \models \varphi$ and $t \models \psi$

$t \models \neg\varphi$ iff not $t \models \varphi$

$t \models \bigcirc \varphi$ iff $t_1 t_2 \ldots \models \varphi$

$t \models \varphi \, U \, \psi$ iff exists $n \geq 0$ s.t.

1. for all $0 \leq i < n$, $t_i t_{i+1} \ldots \models \varphi$
2. $t_n t_{n+1} \ldots \models \psi$

Defined modalities

$\bigcirc$	X	next
U	U	until
$\Diamond \varphi = \text{true U } \varphi$	F	eventually
$\Box \varphi = \neg \Diamond \neg \varphi$	G	always
$\varphi \text{ W} \psi = (\varphi \text{ U } \psi) \vee \Box \varphi$	W	waiting-for (weak-until)

Important properties

Invariance $\Box\ a$

$\Box\ \neg\ (in\_cs1 \wedge in\_cs2)$

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

$\Box\ (\ req\_cs1\ \Rightarrow$

$\neg in\_cs2\ W\ in\_cs2\ W\ \neg in\_cs2\ W\ in\_cs1\ )$

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

$\Box\ (req\_cs1\ \Rightarrow\ \Diamond\ in\_cs1)$

Composed modalities

$\Box\Diamond a$	infinitely often a
$\Diamond\Box a$	almost always a

Where did fairness go ?

Unlike in CTL, fairness can be expressed in LTL !

So there is no need for fairness in the model.

Weak (Buchi) fairness :

$$\neg \Diamond\Box \text{ (enabled} \wedge \neg \text{ taken)}$$

$$\Box\Diamond \text{ (enabled} \Rightarrow \text{ taken)}$$

Strong (Streett) fairness :

$$(\ \Box\Diamond \text{ enabled }) \Rightarrow (\ \Box\Diamond \text{ taken })$$

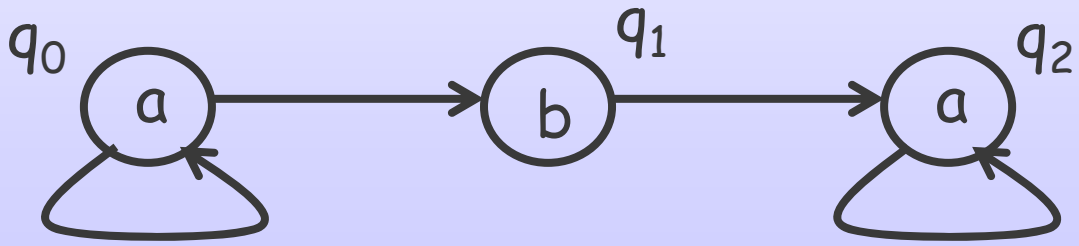
Starvation freedom, corrected

$$\Box\Diamond\,(\text{in\_cs2} \Rightarrow \text{out\_cs2}) \Rightarrow$$
$$\Box\,(\text{req\_cs1} \Rightarrow \Diamond\,\text{in\_cs1})$$

CTL cannot express fairness

$$\forall\Diamond\Box\, a \quad \neq \quad \forall\Diamond\, \forall\Box\, a$$

$$\exists\Box\Diamond\, b \quad \neq \quad \exists\Box\, \exists\Diamond\, b$$



LTL cannot express branching

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

So, LTL and CTL are incomparable.

(There are branching logics that can express fairness, e.g. CTL* = CTL + LTL, but they lose the computational attractiveness of CTL.)