# Temporal Logics & Model Checking

## Formal Methods

## Lecture 4

Farn Wang
Dept. of Electrical Engineering
National Taiwan University

## History of Temporal Logic

- Designed by philosophers to study the way that time is used in natural language arguments
- Reviewed by Prior [PR57, PR67]

- Brought to Computer Science by Pnueli [PN77]
- Has proved to be useful for specification of concurrent systems

## Amir Pnueli
## 1941

- Professor, Weizmann Institute
- Professor, NYU
- Turing Award, 1996

Presentation of a gift at ATVA /FORTE 2005, Taipei

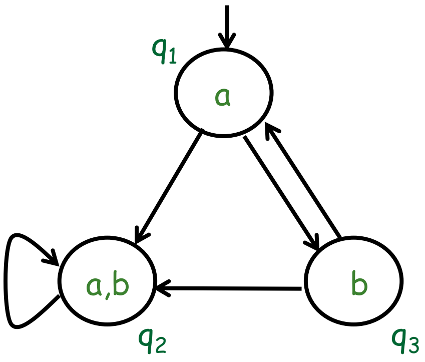
## Kripke structure

A = (S, $S_0$, R, L)

- S
  - a set of all states of the system
- $S_0 \subseteq S$
  - a set of initial states
- $R \subseteq S \times S$
  - a transition relation between states
- $L : R \mapsto 2^P$
  - a function that associates each state with set of propositions true in that state

## Kripke Model

- Set of states S
  - $\{q_1,q_2,q_3\}$
- Set of initial states $S_0$
  - $\{q_1\}$
- Set of atomic propositions AP
  - {a,b}



## Example of Kripke Structure

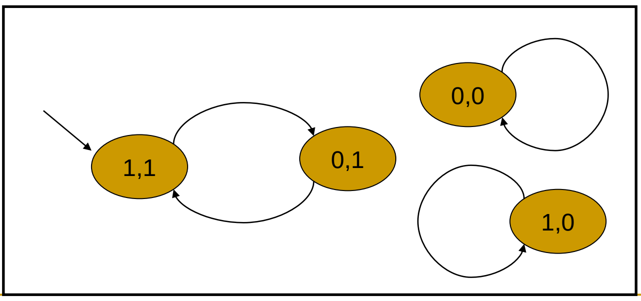
Suppose there is a program

```
initially x=1 and y=1;
while true do
x:=(x+y) mod 2;
endwhile
```

where x and y range over *D*={0,1}

## Example of Kripke Structure

- $S=D\times D$
- $S_0=\{(1,1)\}$
- $R=\{((1,1),(0,1)),((0,1),(1,1)),((1,0),(1,0)),((0,0),(0,0))\}$
- $L((1,1))=\{x=1,y=1\},L((0,1))=\{x=0,y=1\},$ $L((1,0))=\{x=1,y=0\},L((0,0))=\{x=0,y=0\}$



## Fairness

- Interested in the correctness along fair computation paths
- Weak (Büchi) fairness:
  - "an action can not be enabled forever without being taken"
  - necessary for modeling asynchronous models
- Strong (Streett) fairness:
  - "an action can not be enabled infinitely often without being taken"
  - necessary for modeling synchronous interaction

## Framework

- Temporal Logic is a class of Modal Logic
- Allows qualitatively describing and reasoning about changes of the truth values over time
- Usually implicit time representation
- Provides variety of temporal operators (*sometimes, always*)
- Different views of time (branching vs. linear, discrete vs. continuous, past vs. future, etc.)

## Outline

- Linear
  - LPTL (Linear time Propositional Temporal Logics),
  - also called PTL, LTL
- Branching
  - CTL (Computation Tree Logics)
  - CTL* (the full branching temporal logics)

## Temporal Logics：Catalog

| | | |
|---|---|---|
| [✓ propositional] | ↔ | first-order |
| [✓ global] | ↔ | compositional |
| [✓ branching] | ↔ | [✓ linear-time] |
| [✓ points] | ↔ | intervals |
| [✓ discrete] | ↔ | continuous |
| past | ↔ | [✓ future] |

## Temporal Logics

- Linear
  - LPTL (Linear time Propositional Temporal Logics)
- Branching
  - CTL (Computation Tree Logics)
  - CTL* (the full branching temporal logics)

## LPTL (PTL, LTL) Linear-Time Propositional Temporal Logic

Conventional notation：

- propositions : ***p, q, r, …***
- sets : ***A, B, C, D, …***
- states : ***s***
- state sequences : ***S***
- formulas : ***φ,ψ***
- Set of natural number：*N* **= *{0, 1, 2, 3, …}***
- Set of real number：*R*

## LPTL

Given ***P*** : a set of propositions,

a Linear-time structure : *state sequence*

$S= s_0 s_1 s_2 s_3 s_4 \ldots s_k \ldots\ldots$

$s_k$ is a function of P where $s_k : P \rightarrow \{true, false\}$

***or*** $s_k \in 2^P$

example: P={a,b}

{a}{a,b}{a}{a}{b}…

## LPTL
### - syntax

$$\psi ::= \mathbf{true} \mid p \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \bigcirc\psi \mid \psi_1 \mathcal{U} \psi_2$$

abbreviation

| | | |
|---|---|---|
| **false** | $\equiv$ | $\neg$ true |
| $\psi_1 \wedge \psi_2$ | $\equiv$ | $\neg((\neg\psi_1) \vee (\neg\psi_2))$ |
| $\psi_1 \rightarrow \psi_2$ | $\equiv$ | $(\neg\psi_1) \vee \psi_2$ |
| $\Diamond\psi$ | $\equiv$ | $\mathbf{true}\ \mathcal{U}\psi$ |
| $\Box\psi$ | $\equiv$ | $\neg\Diamond\neg\psi$ |

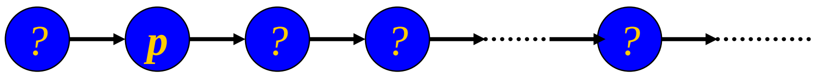
## LPTL
### - syntax

| Exam. | Symbol in CMU | |
|---|---|---|
| $\bigcirc p$ | $\mathbf{X}p$ | ***p is*** **true on next state** |
| $p\mathcal{U}\ q$ | $p\mathcal{U}\ q$ | **From now on, *p* is always true until *q* is true** |
| $\Diamond p$ | $\mathbf{F}p$ | **From now on, there will be a state where *p* is eventually (sometimes) true** |
| $\Box p$ | $\mathbf{G}p$ | **From now on, *p* is always true** |

## LPTL

### - syntax

○*p* **X*p*** ***p* is true on next state**

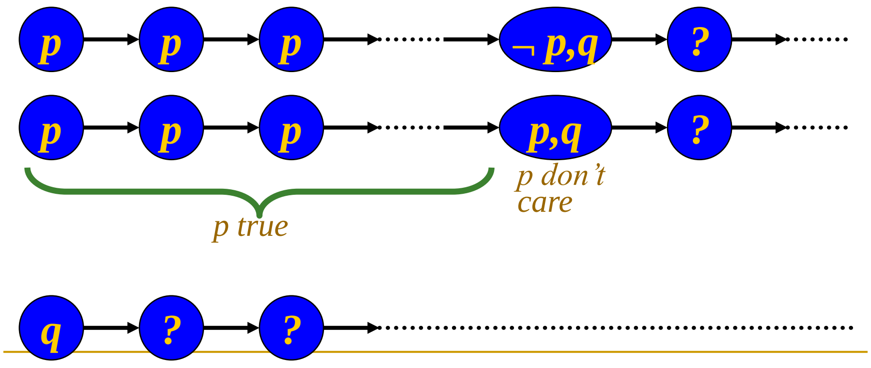


? : don't care

## LPTL

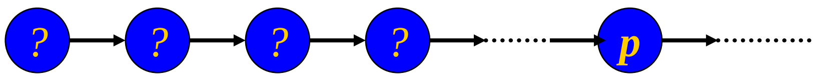
### - syntax

***p*U *q*** ***p****U* ***q*** **From now on, *p* is always true until *q* is true**

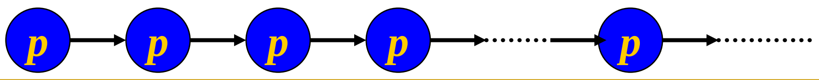


## LPTL
### - syntax

| | | |
|---|---|---|
| $\Diamond p$ | F*p* | **From now on, there will be a state where *p* is eventually (sometimes) true** |



| | | |
|---|---|---|
| $\Box p$ | G*p* | **From now on, *p* is always true** |



## LPTL
### - syntax

Two operator for Fairness

- $\Diamond^{\infty} p \equiv \Box\Diamond p$ **; *p* will happen infinitely many times**
  **infinitely often**

- $\Box^{\infty} p \equiv \Diamond\Box p$ **; *p* will be always true after some time in the future**
  **almost everywhere**

## LPTL
## - semantics

***suffix path :***

$S = s_0\, s_1\, s_2\, s_3\, s_4\, s_5 \ldots\ldots\ldots$

$S^{(0)} = s_0\, s_1\, s_2\, s_3\, s_4\, s_5 \ldots\ldots\ldots$

$S^{(1)} = s_1\, s_2\, s_3\, s_4\, s_5\, s_6 \ldots\ldots\ldots$

$S^{(2)} = s_2\, s_3\, s_4\, s_5\, s_6 \ldots\ldots\ldots$

$S^{(3)} = s_3\, s_4\, s_5\, s_6 \ldots\ldots\ldots$

$S^{(k)} = s_k\, s_{k+1}\, s_{k+2}\, s_{k+3} \ldots\ldots\ldots$

## LPTL
## - semantics

Given a state sequence

$$S = s_0\, s_1\, s_2\, s_3\, s_4 \ldots\, s_k \ldots\ldots$$

We define $S \models \psi$ (S satisfies $\psi$) inductively as：

- $S \models \text{true}$
- $S \models p \Leftrightarrow s_0(p)=\text{true}$, or equivalently $p \in s_0$
- $S \models \neg\psi \Leftrightarrow S \models \psi$ is false
- $S \models \psi_1 \vee \psi_2 \Leftrightarrow S \models \psi_1$ or $S \models \psi_2$
- $S \models \bigcirc\psi \Leftrightarrow S^{(1)} \models \psi$
- $S \models \psi_1 U \psi_2 \Leftrightarrow \exists k \geq 0 (S^{(k)} \models \psi_2 \wedge \forall 0 \leq j < k (S^{(j)} \models \psi_1))$

## LPTL
## - semantics, remarks (1/2)

Basic assumption :

- Isomorphism: *(N，<)*
  - discrete ；suitable for digital computer
  - Initial point（0） ；computer needs reboot
  - Infinite future ；finite and infinite
- Every element in N is a state
  - Every state only have one successor

## LPTL
## - semantics, remarks (2/2)

Example: When memory-fault, generate interrupt

Basic propositions: memf, intr

**j could be in the past ?**

$\forall i \geq 0(\text{memf}(i) \rightarrow \exists j, \text{intr}(j))$

**j is in the past!**

$\forall i \geq 0(\text{memf}(i) \rightarrow \exists j(j<i \wedge \text{intr}(j))$

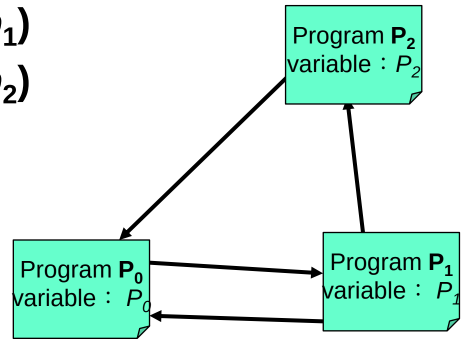
$\forall i \geq 0(\text{memf}(i) \rightarrow \exists j(j>i \wedge \text{intr}(j))$

## LPTL
## - examples (I)(1/6)

$\mathbf{P_0:(p_0:=0 \mid p_0 := p_0 \vee p_1 \vee p_2)}$

$\mathrm{P}_1\mathbf{:(p_1:=0 \mid p_1 := p_0 \vee p_1)}$

$\mathrm{P}_2\mathbf{:(p_2:=0 \mid p_2 := p_1 \vee p_2)}$



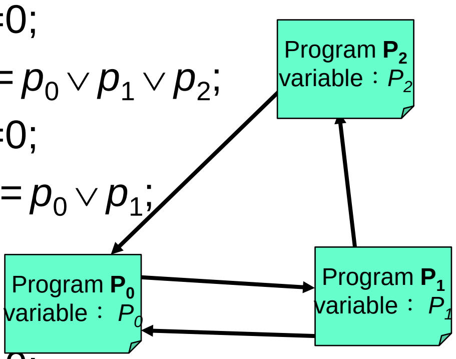
## LPTL
## - examples (I)(2/6)

$\mathrm{P}_0$: when (true) may $p_0$=0;

when *(true) may* $p_0 = p_0 \vee p_1 \vee p_2$;

$\mathrm{P}_1$: when (true) may $p_1$=0;

when (true) may $p_1 = p_0 \vee p_1$;



$\mathrm{P}_2$: when (true) may $p_2$=0;

when (true) may $p_2 = p_1 \vee p_2$;

## LPTL
### - examples(I)(3/6)

$P_0$: when (true) may $p_0$=0;
when *(true) may* $p_0 = p_0 \vee p_1 \vee p_2$;
$P_1$: when (true) may $p_1$=0;
when (true) may $p_1 = p_0 \vee p_1$;
$P_2$: when (true) may $p_2$=0;
when (true) may $p_2 = p_1 \vee p_2$;

$\Box($ (true $\rightarrow \bigcirc\neg p_0$ /*P0*/
$\vee((p_0 \vee p_1 \vee p_2) \rightarrow \bigcirc p_0)$ /*P0*/
$\vee$ true $\rightarrow \bigcirc\neg p_1$ /*P1*/
$\vee ((p_0 \vee p_1) \rightarrow \bigcirc p_1)$ /*P1*/
$\vee$ true $\rightarrow \bigcirc\neg p_2$ /*P2*/
$\vee ((p_1 \vee p_2) \rightarrow \bigcirc p_2)$ /*P2*/
)

But this is a wrong model!

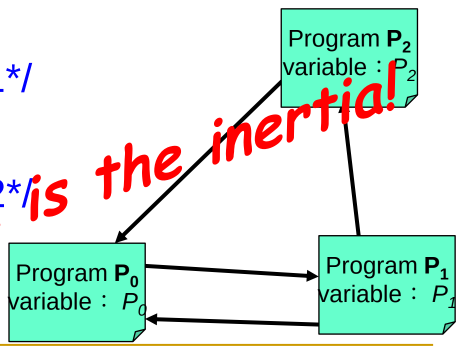
Where is the inertia!

## LPTL
### - examples (I)(4/6)

$\Box($ $(\bigcirc\neg p_0$ /*P0*/
$\vee((p_0 \vee p_1 \vee p_2) \wedge \bigcirc p_0)$ /*P0*/
$\vee \bigcirc\neg p_1$ /*P1*/
$\vee ((p_0 \vee p_1) \wedge \bigcirc p_1)$ /*P1*/
$\vee \bigcirc\neg p_2$ /*P2*/
$\vee ((p_1 \vee p_2) \wedge \bigcirc p_2)$ /*P2*/
)

But this is again a wrong model!

Where is the inertia!

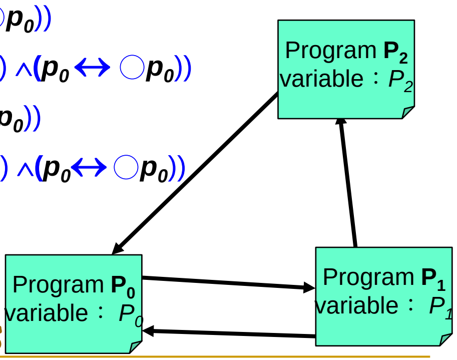

Program $P_2$ variable : $P_2$
Program $P_0$ variable : $P_0$
Program $P_1$ variable : $P_1$

## LPTL

## - examples (I)(5/6)

$P_0$: when (true) may $p_0=0$;
when *(true) may* $p_0 = p_0 \vee p_1 \vee p_2$;
$P_1$: when (true) may $p_1=0$;
when (true) may $p_1 = p_0 \vee p_1$;
$P_2$: when (true) may $p_2=0$;
when (true) may $p_2 = p_1 \vee p_2$;

$$
\begin{aligned}
\Box( \; & (\bigcirc\neg p_0 \wedge (p_1 \leftrightarrow \bigcirc p_1) \wedge (p_2 \leftrightarrow \bigcirc p_2)) \\
\vee \; & (((p_0 \vee p_1 \vee p_2) \leftrightarrow \bigcirc p_0) \wedge (p_1 \leftrightarrow \bigcirc p_1) \wedge (p_2 \leftrightarrow \bigcirc p_2)) \\
\vee \; & (\bigcirc\neg p_1 \wedge (p_2 \leftrightarrow \bigcirc p_2) \wedge (p_0 \leftrightarrow \bigcirc p_0)) \\
\vee \; & (((p_0 \vee p_1) \leftrightarrow \bigcirc p_1) \wedge (p_2 \leftrightarrow \bigcirc p_2) \wedge (p_0 \leftrightarrow \bigcirc p_0)) \\
\vee \; & (\bigcirc\neg p_2 \wedge (p_1 \leftrightarrow \bigcirc p_1) \wedge (p_0 \leftrightarrow \bigcirc p_0)) \\
\vee \; & (((p_1 \vee p_2) \leftrightarrow \bigcirc p_2) \wedge (p_1 \leftrightarrow \bigcirc p_1) \wedge (p_0 \leftrightarrow \bigcirc p_0)) \\
)
\end{aligned}
$$


Program $P_2$ variable : $P_2$
Program $P_0$ variable : $P_0$
Program $P_1$ variable : $P_1$

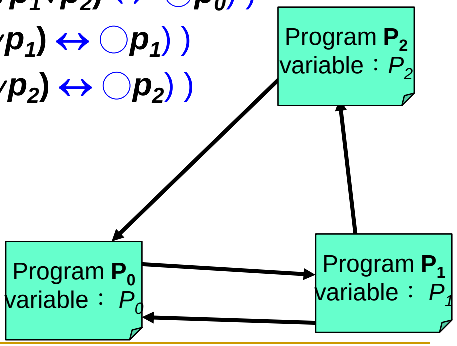
Asynchronous system!
Interleaving semantics

## LPTL

## - examples (I)(6/6)

$P_0$: when (true) may $p_0=0$;
when *(true) may* $p_0 = p_0 \vee p_1 \vee p_2$;
$P_1$: when (true) may $p_1=0$;
when (true) may $p_1 = p_0 \vee p_1$;
$P_2$: when (true) may $p_2=0$;
when (true) may $p_2 = p_1 \vee p_2$;

$$
\begin{aligned}
\Box( \; & ((p_0 \leftrightarrow \bigcirc p_0) \vee \bigcirc\neg p_0 \vee ((p_0 \vee p_1 \vee p_2) \leftrightarrow \bigcirc p_0)) \\
\wedge \; & ((p_1 \leftrightarrow \bigcirc p_1) \vee \bigcirc\neg p_1 \wedge ((p_0 \vee p_1) \leftrightarrow \bigcirc p_1)) \\
\wedge \; & ((p_2 \leftrightarrow \bigcirc p_2) \vee \bigcirc\neg p_2 \vee ((p_1 \vee p_2) \leftrightarrow \bigcirc p_2)) \\
)
\end{aligned}
$$

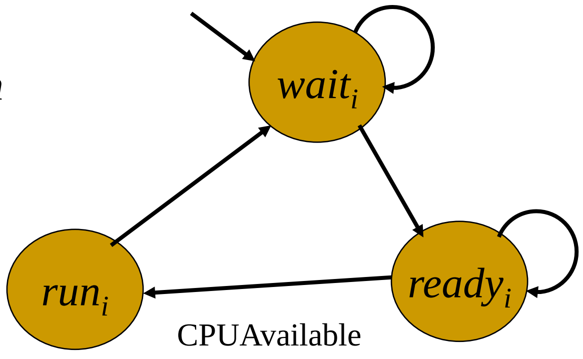

Program $P_2$ variable : $P_2$
Program $P_0$ variable : $P_0$
Program $P_1$ variable : $P_1$

Synchronous

## 2009/12/23 stopped here.

## LPTL
## - examples (II)

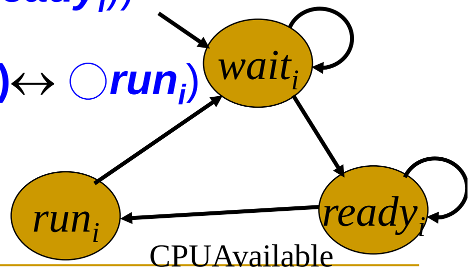
Process$_i$, $1 \leq i \leq m$



Also describe the mutual exclusion condition

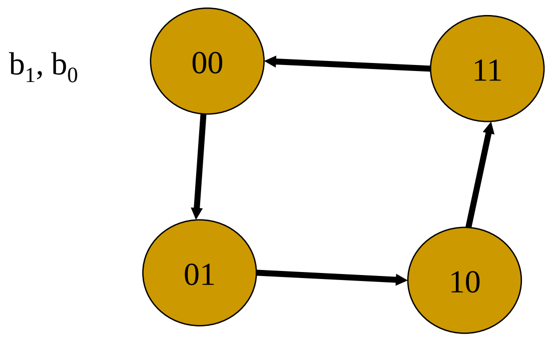
## LPTL
## - examples (II)

$$\wedge_{1\le i\le m}\, wait_i$$
$$\wedge\wedge_{1\le i\le m}\, \Box(run_i \leftrightarrow \wedge_{i<j\le m}\, \neg run_j)$$
$$\wedge\wedge_{1\le i\le m}\, \Box($$
$$(wait_i \leftrightarrow (\bigcirc wait_i \vee \bigcirc ready_i))$$
$$\vee\ (ready_i \leftrightarrow \bigcirc\ ready_i)$$
$$\vee\ ((ready_i \wedge\wedge_{1\le j\le m} \neg\ run_j) \leftrightarrow \bigcirc run_i)$$
$$\vee\ (run_i \leftrightarrow \bigcirc\ wait_i)$$
$$)$$



## LPTL
## - examples (III)

A 2-bit counter operates at bit-level.



## LPTL
## - examples (IV)

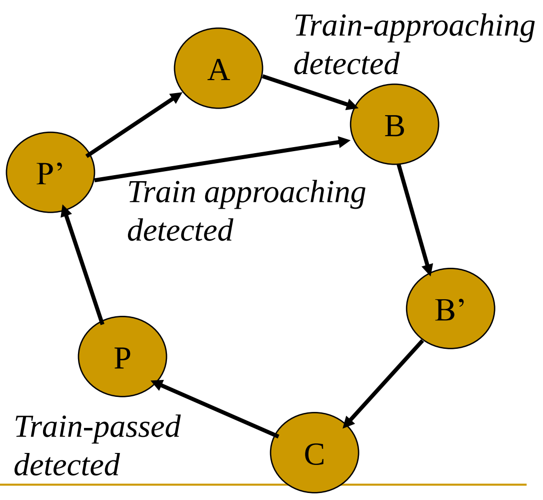
Gate-controller

A: train far-**A**way

B: **B**efore train-crossing

C: train at **C**rossing

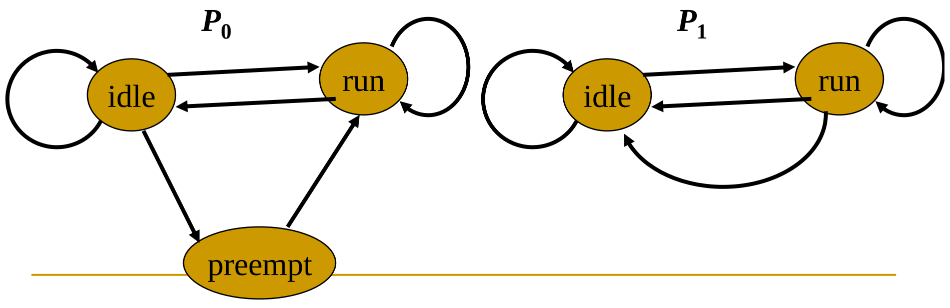
P: train just **P**assed



## LPTL
## - examples (V)

two processes：

$\boldsymbol{P_0}$: high priority; $\boldsymbol{P_1}$: low priority

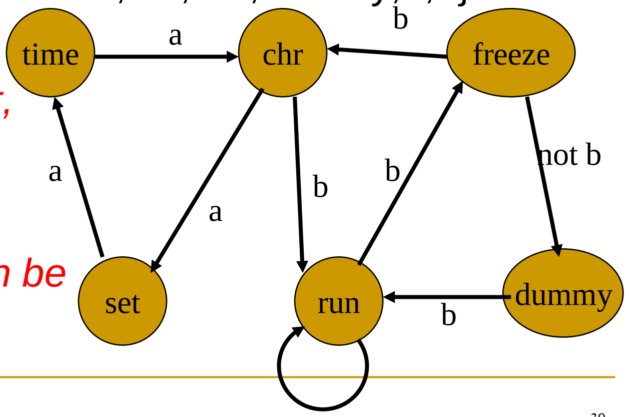


## LPTL
## - examples (VI)

a digital watch：

AP={time,chr,freeze,set,run,dummy,a,b}

- *exactly one of time, chr, freeze, set, run, and dummy can be true at any moment.*



## LPTL
## - workout

Please construct the LPTL formulas for the examples in example III-VI.

## LPTL
## - extensions (1/3)

- ***until*** vs. ***unless***
- ***strict future***
- ***weak***$\bigcirc$ vs. ***strong***$\bigcirc$
- **future vs. past**

$\Diamond^{+}p$ ...................... $\Diamond^{-}p$

$\Box^{+}p$ ...................... $\Box^{-}p$

$\bigcirc^{+}p$ ...................... $\bigcirc^{-}p$

$p\,\mathcal{U}^{+}q$ ...................... $p\,\mathcal{U}^{-}q$ ($p\,\mathcal{S}\,q$)

***since***

## LPTL
## - extensions (2/3)

decidable extension

$\forall i \geq 0(\mathrm{memf}(i) \rightarrow \exists\, j(j>i \wedge j<i+4 \wedge \mathrm{intr}(j))$

undecidable extensions:

- polynomial operations on variables.

$\forall i \geq 0(\mathrm{memf}(i) \rightarrow \exists\, j(j>i+i \wedge \mathrm{intr}(j))$

- 2nd order logics:

$\forall i \geq 0(memf(i) \rightarrow \exists\, f(f(i)>i*i \wedge intr(f(i)))$

## LPTL
## - extensions (3/3)

- First-Order LTL
  - new elements
    - variables, universe, quantifications
    - functions, predicates,
  - interpreted vs. uninterpreted
  - multi-sorted
- Ostroff's RTTL

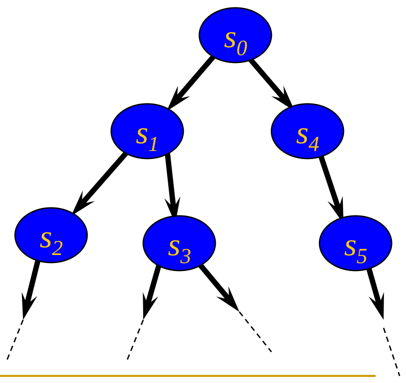
$$\forall x \Box((p \wedge x=T) \rightarrow \exists y \Diamond(q \wedge y=T \wedge y-x<5))$$

## Branching Temporal Logics

Basic assumption of tree-like structure

•Every node is a function of P→{true,false}

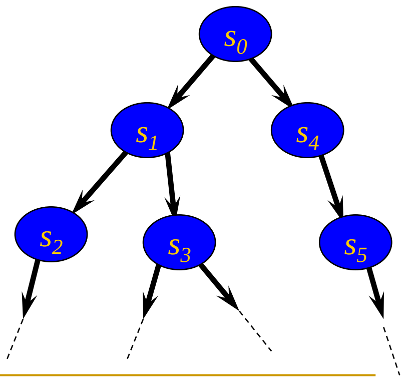
•Every state may have many successors



## Branching Temporal Logics

Basic assumption of tree-like structure

- Every path is isomorphic as ***N***
  - Correspond to a state sequence



Path : $s_0\ s_1\ s_3 \ldots\ldots$

$s_0\ s_1\ s_2 \ldots\ldots$

$s_1\ s_3 \ldots\ldots$

$s_4\ s_5 \ldots\ldots$

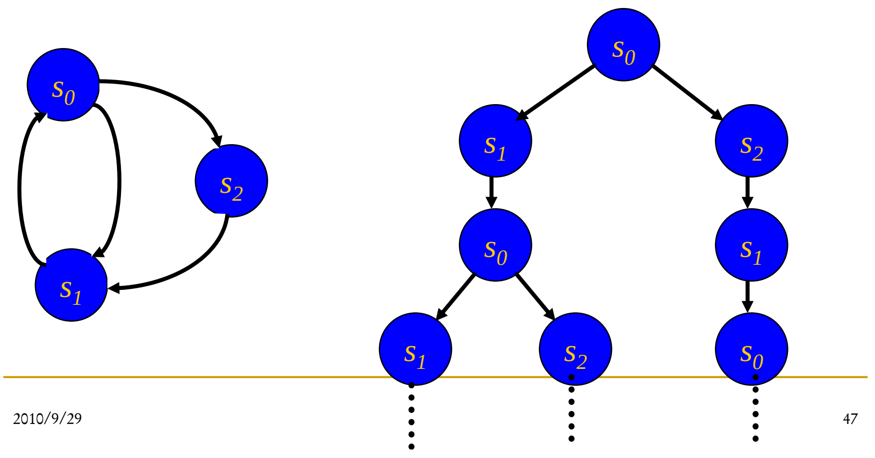
## Branching Temporal Logic

It can accommodate infinite and dense state successors

- In CTL and CTL*, it can’t tell
  - Finite and infinite
    - Is there infinite transitions？
  - Dense and discrete
    - Is there countable（ω）transitions？

## Branching Temporal Logic

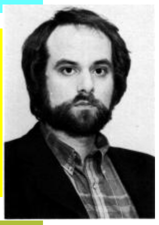
Get by flattening a finite state machine



## CTL(Computation Tree Logic)

Edmund M. Clarke
Professor, CS & ECE
Carnegie Mellon University

E. Allen Emerson
Professor, CS
The University of Texas at Austin



Chin-Laung Lei
Professor, EE
National Taiwan University

## CTL(Computation Tree Logic) - syntax

$\varphi ::= \text{true} \mid p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists\bigcirc\varphi \mid \forall\bigcirc\varphi$
$\mid \exists\varphi_1 U \varphi_2 \mid \forall\varphi_1 U \varphi_2$

abbreviation：

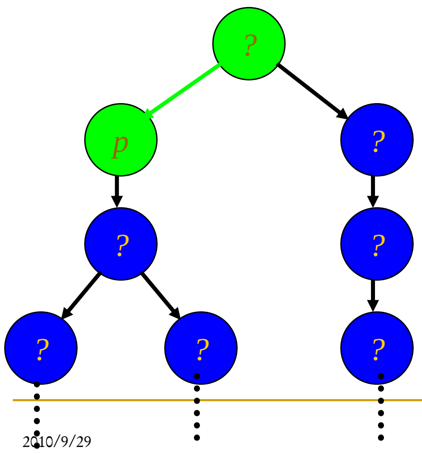
| | | |
|---|---|---|
| false | $\equiv$ | $\neg$ true |
| $\varphi_1 \wedge \varphi_2$ | $\equiv$ | $\neg((\neg\varphi_1) \vee (\neg\varphi_2))$ |
| $\varphi_1 \rightarrow \varphi_2$ | $\equiv$ | $(\neg\varphi_1) \vee \varphi_2$ |
| $\exists\Diamond\varphi$ | $\equiv$ | $\exists$true $U\varphi$ |
| $\forall\Box\varphi$ | $\equiv$ | $\neg\exists\Diamond\neg\varphi$ |
| $\forall\Diamond\varphi$ | $\equiv$ | $\forall$true $U\varphi$ |
| $\exists\Box\varphi$ | $\equiv$ | $\neg\forall\Diamond\neg\varphi$ |

## CTL - semantics

| example | symbol in CMU | |
|---|---|---|
| $\exists\bigcirc p$ | EX*p* | there exists a path where *p* is true on next state |
| $\exists p U q$ | *p*EU*q* | from now on, there is a path where *p* is always true until *q* is true |
| $\forall\bigcirc p$ | AX*p* | for all path where *p* is true on next state |
| $\forall p U q$ | *p*AU*q* | from now on, for all path where *p* is always true until *q* is true |

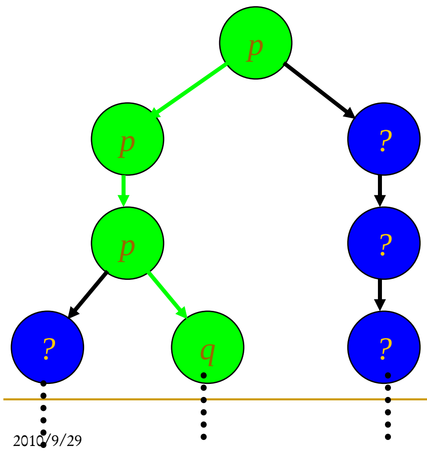
## CTL
- semantics

∃○*p* EX*p* **there exists a path where *p* is true on next state**



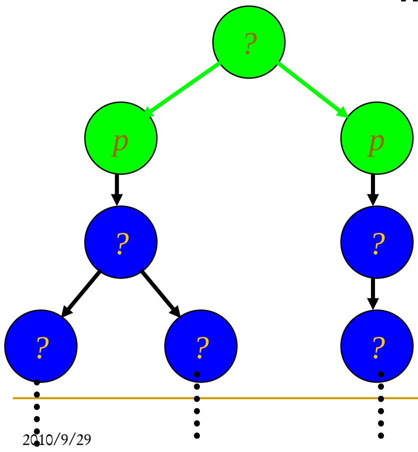
## CTL
- semantics

∃*p*U *q* ***p*EU*q*** **from now on, there is a path where *p* is always true until *q* is true**



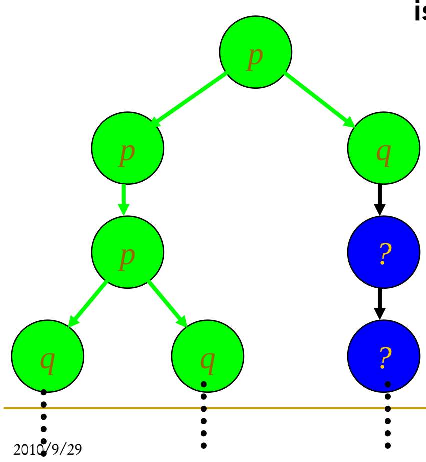
## CTL
- semantics

∀○*p*      **AX*p***      **for all path where *p* is true on next state**



## CTL
- semantics

∀*p*U *q*      ***p*AU*q***      **from now on, for all path where *p* is always true until *q* is true**



## CTL
### - semantic

Assume there are

- a tree stucture ***M***,
- one state ***s*** in ***M*,** and
- a CTL fomula **φ**

***M*,*s*⊨*φ*** means ***s*** in ***M*** satisfy **φ**

## CTL
### - semantics

***s*-path**：a path in *M* that starts from ***s***

$s_0$*-path:*

$s_0\ s_1\ s_2\ s_3\ s_5$ ………
$s_0\ s_1\ s_6\ s_7\ s_8$ ………

$s_1$*-path:*

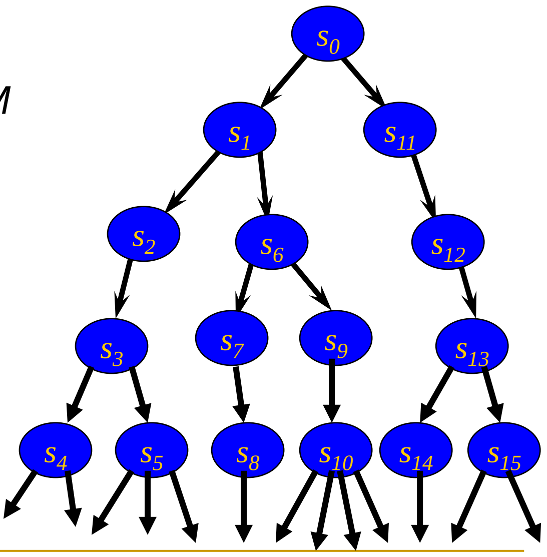
$s_1\ s_2\ s_3\ s_5$ …………

$s_2$*-path:*

$s_2\ s_3\ s_5$ …………

$s_{13}$*-path:*

$s_{13}\ s_{15}$ …………

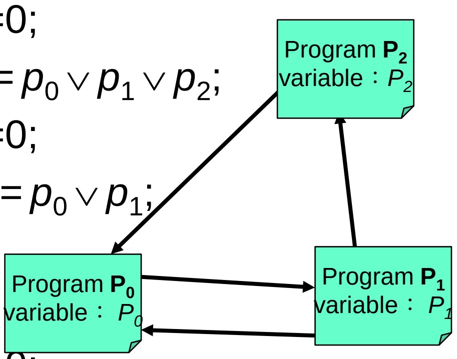


## CTL
### - semantics

- $M,s \vDash \text{true}$
- $M,s \vDash p \iff p \in s$
- $M,s \vDash \neg\varphi \iff$ it is false that $M,s \vDash \varphi$
- $M,s \vDash \varphi_1 \vee \varphi_2 \iff M,s \vDash \varphi_1$ or $M,s \vDash \varphi_2$
- $M,s \vDash \exists\bigcirc\varphi \iff \exists$ s-path $= s_0\ s_1 \ldots\ldots (M,s_1 \vDash \varphi)$
- $M,s \vDash \forall\bigcirc\varphi \iff \forall$ s-path $= s_0\ s_1 \ldots\ldots (M,s_1 \vDash \varphi)$
- $M,s \vDash \exists\varphi_1 U \varphi_2 \iff \exists$ s-path $= s_0\ s_1 \ldots\ldots, \exists k \geq 0$ $(M,s_k \vDash \varphi_2 \wedge \forall 0 \leq j < k (M,s_j \vDash \varphi_1))$
- $M,s \vDash \forall\varphi_1 U \varphi_2 \iff \forall$s-path $= s_0\ s_1 \ldots\ldots, \exists k \geq 0$ $(M,s_k \vDash \varphi_2 \wedge \forall 0 \leq j < k (M,s_j \vDash \varphi_1))$

## LPTL
### - examples (I)(2/6)

$P_0$: when (true) may $p_0$=0;
when *(true) may* $p_0 = p_0 \vee p_1 \vee p_2$;
$P_1$: when (true) may $p_1$=0;
when (true) may $p_1 = p_0 \vee p_1$;

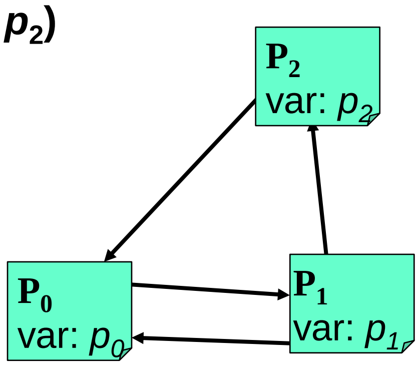


$P_2$: when (true) may $p_2$=0;
when (true) may $p_2 = p_1 \vee p_2$;

## CTL
## - examples (I)

$P_0$:(*$p_0$*:=0 | *$p_0$* := *$p_0$* $\vee$ *$p_1$* $\vee$ *$p_2$*)
$P_1$:(*$p_1$*:=0 | *$p_1$* := *$p_0$* $\vee$ *$p_1$*)
$P_2$:(*$p_2$*:=0 | *$p_2$* := *$p_1$* $\vee$ *$p_2$*)

If $P_0$ is true, it is possible that $P_2$ can be true after the next two cycles.
$\forall\Box(p_0 \rightarrow \exists\bigcirc\ \exists\bigcirc\ p_2)$



## CTL
## - examples (II)

1. If there are dark clouds, it will rain.
   $\forall\Box$(dark-clouds$\rightarrow$ $\forall\Diamond$rain)
2. if a buttefly flaps its wings, the New York stock could plunder.
   $\forall\Box$**(**buttefly-flap-wings $\rightarrow\exists\Diamond$NY-stock-plunder**)**
3. if I win the lottery, I will be happy forever.
   $\forall\Box$(win-lottery $\rightarrow\forall\Box$ happy)
4. In an execution state, if an interrupt occurs in the next cycle, the interrupt handler will execute at the 2nd next cycle.
   $\forall\Box$(exec$\rightarrow\forall\bigcirc$(intrpt$\rightarrow\forall\bigcirc$(intrpt-handler)))

## CTL
### - examples (III)

In an execution state, if an interrupt occurs in the next cycle, the interrupt handler will execute at the 2nd next cycle.

$$\forall\Box(\text{exec}\rightarrow\forall\bigcirc(\text{intrpt}\rightarrow\forall\bigcirc(\text{intrpt-handler})))$$

Some possible mistakes:

$\forall\Box(\text{exec}\rightarrow((\forall\bigcirc\ \text{intrpt})\rightarrow\forall\bigcirc\text{intrpt-handler}))$

$\forall\Box(\text{exec}\rightarrow((\forall\bigcirc\ \text{intrpt})\rightarrow\forall\bigcirc\forall\bigcirc\ \text{intrpt-handler}))$

## CTL*
### - syntax

- CTL* fomula ( state-fomula )

  $\varphi ::= \text{true} \mid p \mid \neg\varphi_1 \mid \varphi_1\vee\varphi_2 \mid \exists\psi \mid \forall\psi$

- path-fomula

  $\psi ::= \varphi \mid \neg\psi_1 \mid \psi_1\vee\psi_2 \mid \bigcirc\psi_1 \mid \psi_1 U \psi_2$

CTL* is set of all state-fomula!

## CTL* - examples (1/4)

In a fair concurrent environment, jobs will eventually finish.

$\forall(((\Box\Diamond \mathit{execute}_1) \wedge (\Box\Diamond \mathit{execute}_2)) \rightarrow \Diamond \mathit{finish})$

**or**

$\forall(((\Diamond^{\infty} \mathit{execute}_1) \wedge (\Diamond^{\infty} \mathit{execute}_2)) \rightarrow \Diamond \mathit{finish})$

## CTL* - examples (2/4)

No matter what, infinitely many comet will hit earth.

$\forall\Box\Diamond$comet-hit-earth

**Or**

$\forall\Diamond^{\infty}$comet-hit-earth

*Why not CTL?*

- $\forall\Box\ \forall\ \Diamond$ comet-hit-earth
- $\forall\Box\ \exists\ \Diamond$ comet-hit-earth

What is the difference ?

## CTL*
## - Workout

- **(1)** ∀☐◇comet-hit-earth
- **(2)** ∀☐ ∀ ◇ comet-hit-earth
- **(3)** ∀☐ ∃ ◇ comet-hit-earth

Please draw trees that tell

- (1) from (2) and (3)
- (2) from (1) and (3)
- (3) from (1) and (2)

## CTL*
## - examples (3/4)

If you never have a lover, I will marry you.

∀**((**☐you-have-no-lover**)** → ◇marry-you**)**

*Why not CTL ?*

- **(**∀☐ you-have-no-lover**)** → ∀ ◇你嫁給我
- **(**∀☐ you-have-no-lover**)** → ∃ ◇你嫁給我
- **(**∃☐ you-have-no-lover**)** → ∀ ◇你嫁給我

## CTL*
## - Workout

- (1)$\forall((\Box$you-have-no-lover$) \rightarrow \Diamond$marry-you$)$
- (2) $(\forall\Box$ you-have-no-lover$) \rightarrow \forall \Diamond$ marry-you
- (3) $(\forall\Box$ you-have-no-lover$) \rightarrow \exists \Diamond$ marry-you
- (4) $(\exists\Box$ you-have-no-lover$) \rightarrow \forall \Diamond$ marry-you

***Please draw trees that tell***

- ***(1) from (2), (3), (4)***
- ***(2) from (1), (3), (4)***
- ***(3) from (1), (2), (4)***
- ***(3) from (1), (2), (3)***

## CTL*
## - examples (4/4)

If I buy lottory tickets infinitely many times, eventually I will win the lottery.

$\forall((\Box\Diamond$buy-lottery$) \rightarrow \Diamond$win-lottery$)$

**or**

$\forall ((\Diamond^{\infty}$buy-lottery$) \rightarrow \Diamond$ win-lottery$)$

## CTL*
### - semantics

***suffix path :***

$S = s_0 s_1 s_2 s_3 s_5 \ldots\ldots\ldots$

$S^{(0)} = s_0 s_1 s_2 s_3 s_5 \ldots\ldots\ldots$

$S^{(1)} = s_1 s_2 s_3 s_5 \ldots\ldots\ldots$

$S^{(2)} = s_2 s_3 s_5 \ldots\ldots\ldots$

$S^{(3)} = s_3 s_5 \ldots\ldots\ldots$

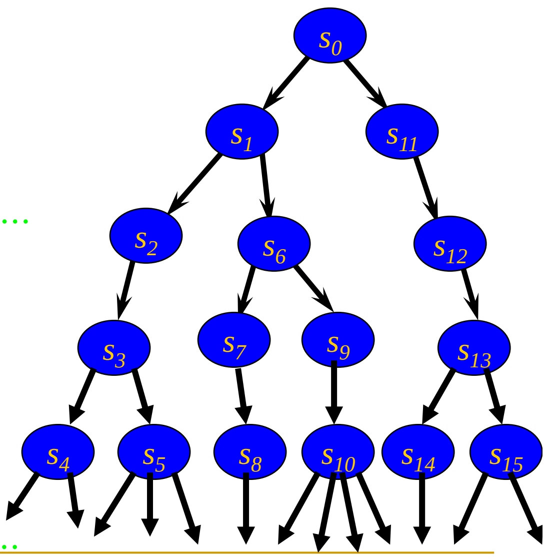
$S^{(4)} = s_5 \ldots\ldots\ldots$

$S = s_0 s_1 s_6 s_7 s_8 \ldots\ldots\ldots$

$S^{(2)} = s_6 s_7 s_8 \ldots\ldots\ldots$

$S = s_0 s_{11} s_{12} s_{13} s_{15} \ldots\ldots\ldots$

$S^{(3)} = s_{13} s_{15} \ldots\ldots\ldots$



## CTL*
### - semantics

*state*-fomula

$\varphi ::= \text{true} \mid p \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \exists\psi \mid \forall\psi$

- $M,s \models \text{true}$
- $M,s \models p \Leftrightarrow p \in s$
- $M,s \models \neg\varphi \Leftrightarrow M,s \models \varphi$ 是false
- $M,s \models \varphi_1 \vee \varphi_2 \Leftrightarrow M,s \models \varphi_1$ or $M,s \models \varphi_2$
- $M,s \models \exists\psi \Leftrightarrow \exists$ s-path = S ($S \models \psi$)
- $M,s \models \forall\psi \Leftrightarrow \forall$ s-path = S ($S \models \psi$)

## CTL*
## - semantics

*path*-fomula

$\psi ::= \varphi \mid \neg\psi_1 \mid \psi_1 \vee \psi_2 \mid \bigcirc\psi \mid \psi_1 U \psi_2$

- *If* $S = s_0 s_1 s_2 s_3 s_4 \ldots\ldots\ldots, S \models \varphi \Leftrightarrow M, s_0 \models \varphi$
- $S \models \neg\psi_1 \Leftrightarrow S \models \psi_1$ 是false
- $S \models \psi_1 \vee \psi_2 \Leftrightarrow S \models \psi_1$ or $S \models \psi_1$
- $S \models \bigcirc\psi \Leftrightarrow S^{(1)} \models \psi$
- $S \models \psi_1 U \psi_2 \Leftrightarrow \exists k \geq 0 \, (S^{(k)} \models \psi_2 \wedge \forall 0 \leq j < k (S^{(j)} \models \psi_1))$

## Expressiveness

Given a language $\mathcal{L}$,

- what model sets $L$ can express ?
- what model sets L cannot ?

model set: a set of behaviors

A formula = a set of models (behaviors)

- for any$\varphi \in \mathcal{L}$, $[\varphi] \overset{\text{def}}{=} \{M \mid M \models \varphi\}$

A language = a set of formulas.

Expressiveness: Given a model set $F$,

$F$ is expressible in $\mathcal{L}$ iff $\exists \varphi \in \mathcal{L}([\varphi] = F)$

## Expressiveness

### Comparison in expressiveness:

Given two languages $L_1$ and $L_2$

Definition: $L_1$ is ***more expressive than*** $L_2$ ($L_2 < L_1$)
iff $\forall \varphi \in \mathcal{L}_2$ **([φ] is expressible in $\mathcal{L}_1$)**

Definition: $L_1$ and $L_2$ ***are expressively equivalent***
**($L_1 \equiv L_2$)** iff **$(L_2 < L_1) \wedge (L_1 < L_2)$**

Definition: **$L_1$ 、 $L_2$ are *expressively incomparable*** iff
**$\neg((L_2 < L_1) \vee (L_1 < L_2))$**

## Expressiveness

- expressiveness of PLTL
  - PLTL & PLTLB
  - PLTL & QPLTL
  - FOLLO & SOLLO
  - regular languages
- expressiveness of branching-time logics

## Expressiveness - LPTL

- PLTL with only future modal operators
- PLTLB with both past and future modal operators

| | | |
|---|---|---|
| $\Diamond^{+}p$ | ..................... | $\Diamond^{-}p$ |
| $\Box^{+}p$ | ..................... | $\Box^{-}p$ |
| $\bigcirc^{+}p$ | ..................... | $\bigcirc^{-}p$ |
| $p\mathcal{U}^{+}q$ | ..................... | $p\mathcal{U}^{-}q$ ($p\mathcal{S}\, q$) |

**Theorem：** *PLTL & PLTLB have the same expressiveness.*

## Expressiveness - LPTL

$\Diamond^{+}(eat \wedge \Diamond^{+}(shit \wedge \Diamond^{-}full))$ **in PLTLB**

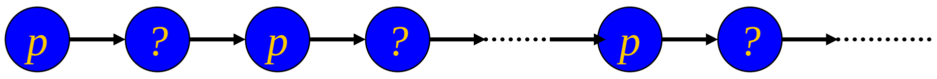
$$\Diamond^{+}(eat \wedge \Diamond^{+}(shit \wedge full)) \quad \text{in PLTL}$$
$$\vee\ \Diamond^{+}(eat \wedge \Diamond^{+}(full \wedge \Diamond^{+} shit))$$
$$\vee\ \Diamond^{+}(full \wedge \Diamond^{+}(eat \wedge \Diamond^{+} shit))$$

**partial-order → total-order**
**PLTL is less succinct than PLTLB.**

## Expressiveness - LPTL

Theorem:

Given $P=\{p\}$, PLTL cannot express the following model.

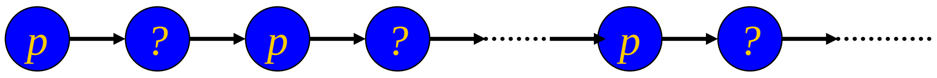


*p is true at only even states.* [P.Wolper 1993]

## Expressiveness - QPTL

QPLTL (Quantified PLTL) can express the following model.

$\exists x(x \wedge (\Box(x \rightarrow \bigcirc\neg x)) \wedge (\Box((\neg x) \rightarrow \bigcirc x)) \wedge (\Box(x \rightarrow p)))$



*p is true at only even states.* [P.Wolper 1993]

***With an auxiliary* proposition *x*,**

***x* initially true.**

***x* alternates from a state to the next.**

***x* → *p***

## Expressiveness - QPTL

QPLTL, syntax

$\psi$ ::= true | p | $\neg\psi$ | $\psi_1 \vee \psi_2$ | $\bigcirc\psi$ | $\psi_1 \mathcal{U} \psi_2$ | $\exists x\psi$

abbreviation:

$$\forall x\psi \equiv \neg\exists x \neg\psi$$

QPLTL, intuitive semantics

- $\exists x\psi$: there is an x-extended state sequence $\models\psi$
- $\forall x\psi$: all x-extended state sequence $\models\psi$

## Expressiveness - QPTL

QPLTL, semantics

Given state sequence $S = s_0\, s_1\, s_2\, s_3\, s_4 \ldots\, s_k \ldots\ldots$

S $\models \exists x\psi$ if and only if

$\exists$ $T = t_0\, t_1\, t_2\, t_3\, t_4 \ldots\, t_k \ldots\ldots$ such that

- $\forall$ $k \geq 0$, $t_k$ is identical to sk except on $t_k(x)$
- T $\models\psi$

## Expressiveness
## - FOLLO

FOLLO (First-Order Language of Linear Order)

- used to define PLTL.
- syntax elements: ℕ, <, ***p*(*i*), ¬, ∨, ∃,** ∀
  - ∃, ∀: quantification over ℕ
  - ***p*(*i*):** monadic predicates of ℕ

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 …… |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***p*** | ***p*** | **¬*p*** | ***p*** | **¬*p*** | **¬*p*** | **¬*p*** | ***p*** | ***p*** | **¬*p*** | ***p*** | ***p*** | ***p*** …… |
| ***q*** | **¬*q*** | ***q*** | ***q*** | ***q*** | **¬*q*** | ***q*** | **¬*q*** | ***q*** | **¬*q*** | ***q*** | ***q*** | ***q*** …… |

## Expressiveness
## - SOLLO

SOLLO(Second-Order Language of Linear Order)

- syntax elements: ℕ, **<**, ***p*(*i*)**, ¬, ∨, ∃, ∀
- ∃, ∀: quantification over
  - *i* ∈ ℕ and
  - x ∈ ℕ ◊{*true*,*false*}

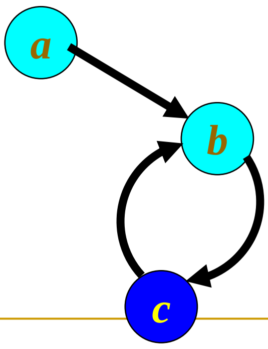
Theorem:

**PLTL≡PLTLB≡FOLLO<SOLLO≡QPLTL≡QPLTLB**

## Expressiveness
## - regular languages

Regular Languages

- recognizable with finite-state automata



***abc***

***abcbc***

***abcbcbc***

***abcbcbc……bc***

***Note***: each a, b, c is encoded with an array of bits.

## Expressiveness
## - regular languages

Regular Languages

- recognizable with finite-state automata

Grammar rules : concatenate, +, *, ¬

| | |
|---|---|
| ***a(bc)**** | ***a(b+c)**** |
| ***a*** | ***a*** |
| ***abc*** | ***ab*** |
| ***abcbcbc*** | ***accc*** |
| ***abcbc……bc*** | ***abbccc……b*** |

## Expressiveness - regular languages

Regular Languages

- recognizable with finite-state automata

Grammar rules : concatenate, +, *, ¬

*a*¬((*b*+*c*)*) assume Σ={a,b,c}

***aa***

***aabbba***

***abcbaaccc***

***a…bacc……***

## Expressiveness - regular languages

How to use PLTL to specify regular languages ?

With **P={p,q}**

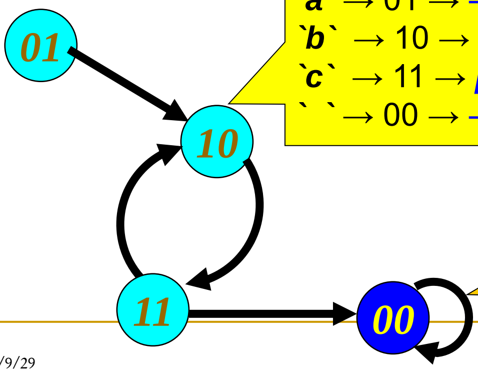
***Encode input symbols with state propositions.***

**\`a\`** → 01 → ¬ ***p*** ∧ **q**

**\`b\`** → 10 → ***p*** ∧ ¬ **q**

**\`c\`** → 11 → ***p*** ∧ **q**

**\` \`** → 00 → ¬ ***p*** ∧ ¬ **q**



Padding an infinite sequence of spaces to each finite words.

## Expressiveness
## - regular languages

The following four are equivalent in expressiveness.

- PLTL
- FOLLO
- regular languages without *
- languages recognizable with counter-free automata.

counter automata: there exists $s_0, s_1, s_2, \ldots, s_{k-1}$ and $w$ such that $s_{i+1} \bmod k \in \delta(s_i, w)$

## Expressiveness
## - regular languages

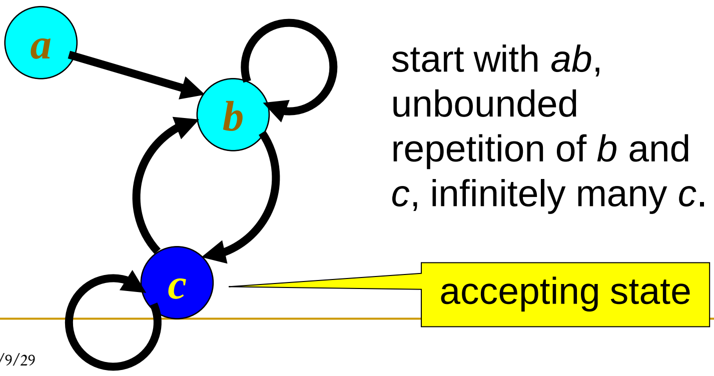
The following four are equivalent in expressiveness.

- QPLTL
- SOLLO
- regular language
- languages recognizable with finite-state automata.

## Expressiveness - regular languages for infinite behaviors

automata accepting infinite strings

- *Büchi accepting: accepting states must appear infinitely many times.*



start with *ab*, unbounded repetition of *b* and *c*, infinitely many *c*.

## Expressiveness - regular languages for infinite behaviors

2 regular languages for infinite strings

- $\alpha(\beta)^\omega$ specifies

  $w_0\, w_1\, w_2\, w_3\, w_4 \ldots w_k \ldots\ldots$

  $w_0 \in \alpha$ and $w_k \in \beta$, for each k> 0

- $\alpha\lim\beta$ specifies

  $a_0\, a_1\, a_2\, a_3\, a_4 \ldots a_k \ldots\ldots$

  with infinitely many k>0

  such that $a_0\, a_1\, a_2\, a_3\, a_4 \ldots a_k \in \alpha\beta$

## Expressiveness
## - regular languages for infinite behaviors

The following four are equivalent in expressiveness.

- PLTL
- FOLLO
- $\cup_{i=1}^{m} \alpha_i \lim\beta_i$
- $\cup_{i=1}^{m} ( \lim \alpha_i \cap \neg\lim \beta_i)$

$\alpha_i$ and $\beta_i$ are regular expressions without *-expressions.

## Expressiveness
## - regular languages for infinite behaviors

The following four are equivalent in expressiveness.

- QPLTL
- SOLLO
- $\cup_{i=1}^{m} \alpha_i (\beta_i^{\omega})$
- $\cup_{i=1}^{m} \alpha_i \lim\beta_i$
- $\cup_{i=1}^{m} ( \lim \alpha_i \cap \neg\lim \beta_i)$

$\alpha_i$ and $\beta_i$ are regular expressions without *-expressions.

## 091230 stopped here.

## Expressiveness
## - branching-time logics

What to compare with ?

- finite-state automata on infinite trees.
- 2nd-order logics with monadic prdicate and many successors (SnS)
- 2nd-order logics with monadic and partial-order

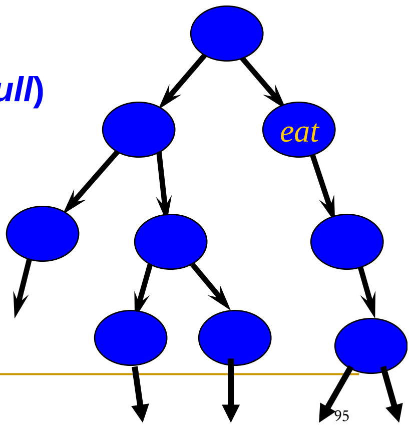
*Very little known at the moment,*

the fine difference in semantics of branching-structures

## Expressiveness
## - CTL*, example (I)

A tree the distinguishes the following two formulas.

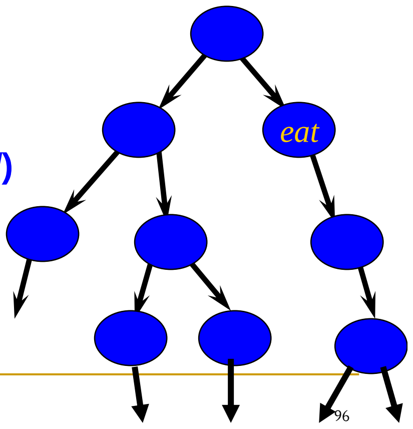
- $\forall((\Diamond eat) \rightarrow \Diamond full)$
  - Negation: $\exists((\Diamond eat) \wedge \Box\neg full)$
- $(\forall\Diamond eat) \rightarrow (\forall\Diamond full)$



## Expressiveness
## - CTL*, example (II)

A tree that distinguishes the following two formulas.

- $\forall((\Box eat) \rightarrow \Diamond full)$
- $\forall\Box\ (eat \rightarrow \forall\Diamond full)$
  - Negation: $\exists\Diamond(eat \wedge\exists\Diamond\neg full)$



## Expressiveness - CTL*

With the abundant semantics in CTL*, we can compare the subclasses of CTL*.

With restrictions on the modal operations after ∃, ∀, we have many CTL* subclasses.

**Example:**

B(¬,∨,○,***U***) : only ¬,∨,○,***U*** *after* ∃, ∀

B(¬,∨,○,◇$^\infty$): only ¬,∨,○,◇$^\infty$ after ∃, ∀

B(○,◇) : only ○,◇after ∃, ∀

## Expressiveness - CTL*

CTL* subclass expressiveness heirarchy

| | | |
|---|---|---|
| CTL* | > | B(¬,∨,○,◇,*U*,◇$^\infty$) |
| | > | B(○,◇,*U*,◇$^\infty$) |
| | > | B(¬,∨,○,◇,*U*) |
| | = | B(○,◇,*U*) |
| | > | B(¬,∨,○,◇) |
| | > | B(○,◇) |
| | > | B(◇) |

## Expressiveness
## - CTL*

Theorem : $B(\neg,\vee,\bigcirc,\Diamond,U) \equiv B(\bigcirc,\Diamond,U)$

Proof: reduction of formulas from $B(\neg,\vee,\bigcirc,\Diamond,U)$ to $B(\bigcirc,\Diamond,U)$.

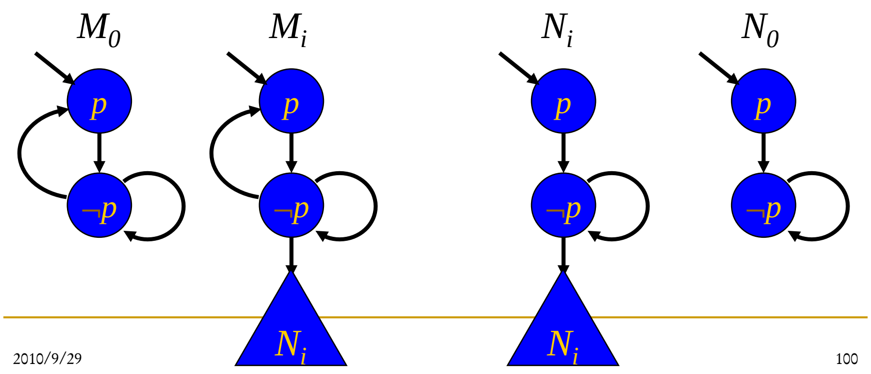
Suppose we have a modality $\exists\psi$ *with* $\psi$ *in DNF and* '$\neg$' only before $U$. (feasible since $\neg\bigcirc\psi_3 \equiv \bigcirc\neg\psi_3$)

- reduce $\neg(\psi_1 U \psi_2)$ to $((\neg\psi_2) U \neg(\psi_2 \wedge \psi_1)) \vee \Box\neg\psi_2$
- reduce $(\psi_1 U \psi_2) \wedge (\psi_3 U \psi_4)$ to
  $((\psi_1 \wedge \psi_3) U (\psi_2 \wedge \exists(\psi_3 U \psi_4))) \vee ((\psi_3 \wedge \psi_1) U (\psi_4 \wedge \exists(\psi_1 U \psi_2)))$
- reduce $(\psi_1 U \psi_2) \wedge \Box\psi_3$ to $(\psi_1 \wedge \psi_3) U (\psi_2 \wedge \exists\Box\psi_3)$
- reduce $\exists(\psi_1 \vee \psi_2 \vee \ldots \vee \psi_n)$ to $(\exists\psi_1) \vee (\exists\psi_2) \vee \ldots \vee (\exists\psi_n)$
- reduce $\exists((\psi_1 U \psi_2) \wedge \bigcirc\psi_3)$ to
  $(\psi_2 \wedge \exists\bigcirc\psi_3) \vee (\psi_1 \wedge \exists\bigcirc(\psi_3 \wedge (\psi_1 U \psi_2)))$

## Expressiveness
## - CTL*

Theorem : $\exists\Diamond^\infty p$ **is inexpressible in** $B(\bigcirc,\Diamond,U)$ **.**

Proof: **induction on** $i$ **: for any** $\varphi \in B(\bigcirc,\Diamond,U)$ **,**
**when** $i>|\varphi|$**,** $\varphi$ cannot distinguish $M_i$ **from** $N_i$ **.**



## Workout

Please complete the proof in details in the previous page.

## Expressiveness - CTL*

Comparing PLTL with CTL*

assumption, all $\varphi \in$ PLTL are interpreted as $\forall\varphi$

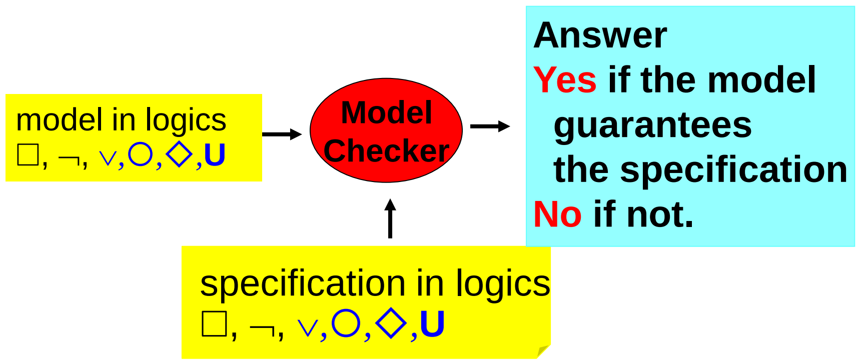
*Intuition:* PLTL is used to specify all runs of a system.

$$\begin{tikzcd}
& B(\neg,\vee,\bigcirc,\Diamond,\mathcal{U},\Diamond^{\infty}) \arrow[dr, ">"] & \\
\text{CTL*} \arrow[ur, ">"] \arrow[dr, ">"'] & & \text{PLTL(F)} \\
& \text{PLTL} \arrow[ur, ">"'] &
\end{tikzcd}$$

## Verification

- LPTL, validity checking $\varphi \vDash \phi$
  - instead, check the satisfiability of $\varphi \wedge \neg\phi$
  - construct a tabelau for $\varphi \wedge \neg\phi$
- model-checking $M \vDash \phi$
  - LPTL: M: a Büchi automata, $\phi$: an LPTL formula
  - CTL: M: a finite-state automata, $\phi$: a CTL formula
- simulation & bisimulation checking $M \vDash M'$

## Satisfiability-checking framework


model in logics
$\Box, \neg, \vee, \bigcirc, \Diamond, \mathbf{U}$
Model Checker
Answer
Yes if the model guarantees the specification
No if not.
specification in logics
$\Box, \neg, \vee, \bigcirc, \Diamond, \mathbf{U}$

## LPTL - tableau for satisfiability checking

Tableau for $\varphi$

- a finite Kripke structure that fully describes the behaviors of $\varphi$
- exponential number of states
- An algorithm can explore a fulfilling path in the tableau to answer the satisfiability.
  - nondeterministic
  - without construction of the tableau
  - PSPACE.

## LPTL - tableau for satisfiability checking

Tableau construction

a preprocessing step: push all negations to the literals.

- $\neg(\psi_1 \wedge \psi_2) \equiv (\neg \psi_1) \vee (\neg \psi_2)$
- $\neg(\psi_1 \vee \psi_2) \equiv (\neg \psi_1) \wedge (\neg \psi_2)$
- $\neg \bigcirc \psi \equiv \bigcirc \neg \psi$
- $\neg\neg \psi \equiv \psi$
- $\neg(\psi_1 \mathbf{U} \psi_2) \equiv (\Box \neg \psi_2) \vee ((\neg \psi_2) \mathbf{U} ((\neg \psi_1) \wedge (\neg \psi_2)))$
- $\neg \Box \psi \equiv \Diamond \neg \psi$

## LPTL
## - tableau for satisfiability checking

Tableau construction

CL($\varphi$) (closure) is the smallest set of formulas containing $\varphi$ with the following consistency requirement.

- $\neg p \in CL(\varphi)$ iff $p \in CL(\varphi)$
- If $\psi_1 \vee \psi_2$ , $\psi_1 \wedge \psi_2 \in CL(\varphi)$, then $\psi_1, \psi_2 \in CL(\varphi)$
- If $\bigcirc\ \psi \in CL(\varphi)$, then $\psi \in CL(\varphi)$
- If $\psi_1 \mathbf{U} \psi_2 \in CL(\varphi)$, then $\psi_1$ , $\psi_2$ , $\bigcirc\ (\psi_1\ \mathbf{U}\ \psi_2\ ) \in CL(\varphi)$
- If $\Box\ \psi \in CL(\varphi)$, then $\psi, \bigcirc\ \Box\ \psi \in CL(\varphi)$
- If $\Diamond\psi \in CL(\varphi)$, then $\psi, \bigcirc\Diamond\psi \in CL(\varphi)$

## LPTL
## - tableau for satisfiability checking

Tableau (V, E), *node consistency condition*:

A tableau node $v \in V$ is a set $v \subseteq CL(f)$ such that

- $p \in v$ iff $\neg p \notin v$
- If $\psi_1 \vee \psi_2 \in v$, then $\psi_1 \in v$ or $\psi_2 \in v$
- If $\psi_1 \wedge \psi_2 \in v$, then $\psi_1 \in v$ and $\psi_2 \in v$
- if $\Box\psi \in v$, then $\psi \in v$ and $\bigcirc\ \Box\ \psi \in v$
- if $\Diamond\psi \in v$, then $\psi \in v$ or $\bigcirc\ \Diamond\ \psi \in v$
- if $\psi_1 \mathbf{U} \psi_2 \in v$, then $\psi_2 \in v$ or ($\psi_1 \in v$ and $\bigcirc\ (\psi_1 \mathbf{U} \psi_2) \in v$)

## LPTL
## - tableau for satisfiability checking

Tableau (V, E), *arc consisitency condition*:

Given an arc (v,v′) ∈E, if ○ ψ ∈ v, then ψ ∈ v′

- A node v in (V,E) is initial for φ if φ ∈v.

## LPTL
## - tableau for satisfiability checking

CL(pUq) = {pUq, ○pUq, p, ¬ p, q, ¬ q }

Exam

tablea

**Workout:**
**Please draw the tableau with arc connections!**

V: q} {p, q}

¬q, pUq, ○ Uq} {p, ¬q}

{¬p, q, pUq, ○pUq} pUq} {¬p,q}

{¬p, q, ○pUq}

{¬p, ¬q, ○pUq} {¬p, ¬q}

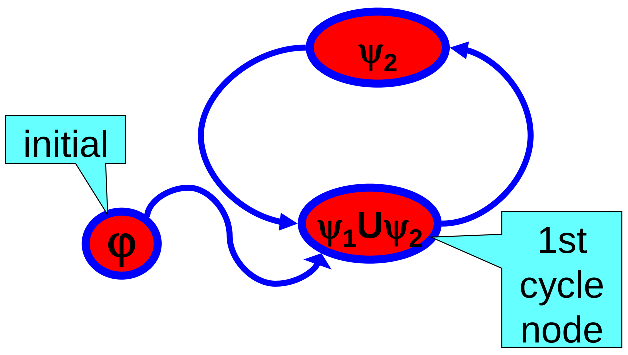
E: ?

## LPTL
## - tableau for satisfiability checking

$\varphi$ is satisfiable iff in (V,E),

- there is an infinite path from an initial node for $\varphi$ such that all until formulas are eventually satisfied; or
- there is a strong connected component (SCC) reachable from an initial node for $\varphi$ such that for all until formula $\psi_1 \mathbf{U} \psi_2$ in a node in the SCC, there is also a node in the SCC containing $\psi_2$ ; or
- there is a cycle reachable from an initial node for $\varphi$ such that the for all until formulas $\psi_1 \mathbf{U} \psi_2$ in **the first cycle node**, there is also a node in the cycle containing $\psi_2$ .

## LPTL
## - tableau for satisfiability checking

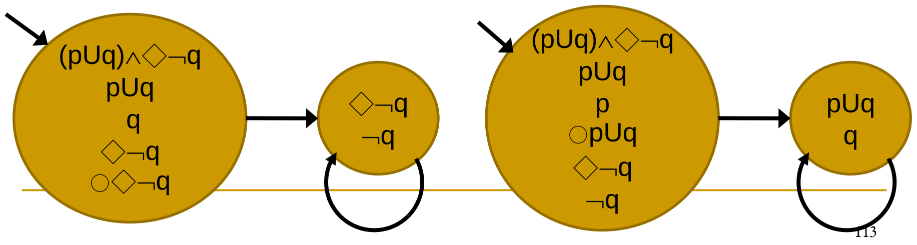


## LPTL
## - tableau for satisfiability checking

Please use tableau method to show that
 pUq ⊨ □q is false.

1) Convert to negation: (pUq)∧◇¬q

2) CL((pUq)∧◇¬q)
 = {(pUq)∧◇¬q, pUq, ○pUq, p, q, ◇¬q, ○◇¬q }



## LPTL
## - tableau for satisfiability checking

Please use tableau method to show that
 pUq ⊨ ◇q is true.

1) Convert to negation: (pUq)∧□¬q

2) CL((pUq)∧□¬q)
 = {(pUq)∧□¬q, pUq, ○pUq, p, q, □¬q, ○□¬q }

Pf: In each path that is a model of (pUq)∧□¬q, q must always be satisfied. Thus, pUq is never fulfilled in the model.

QED

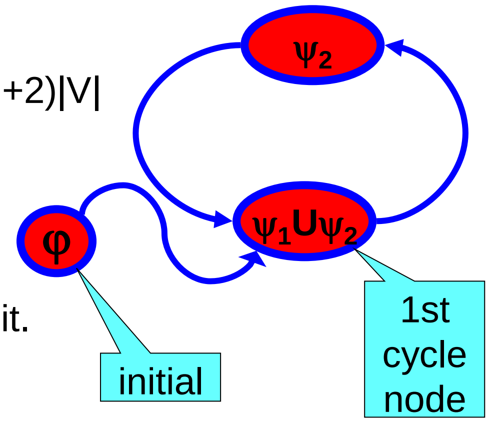
## LPTL

### - tableau for satisfiability checking

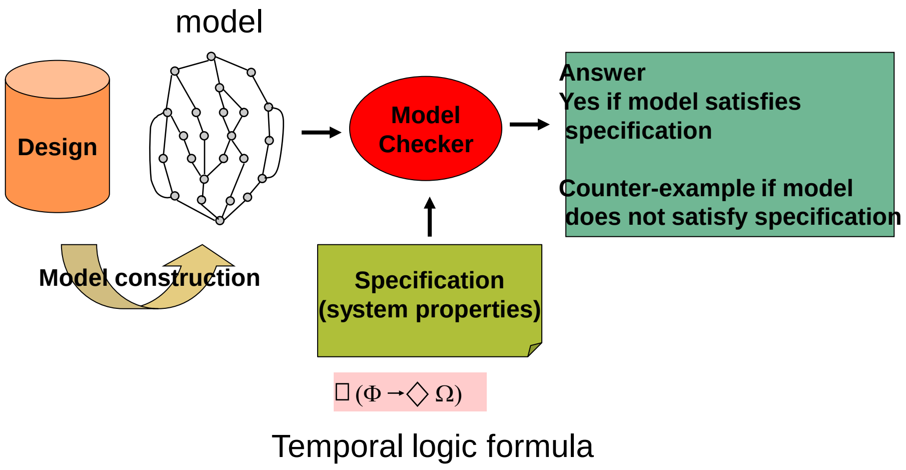
φ is satisfiable iff in (V,E),

there exists ...

- path+cycle≤ (|CL(φ)|+2)|V|
- |CL(φ)| flags to check the until-formulas from the first cycle node.
- nondeterministic PSPACE can solve it.
- PSPACE-complete.



## Model Checking Framework



model
Design
Model construction
Model Checker
Answer
Yes if model satisfies specification
Counter-example if model does not satisfy specification
Specification (system properties)
□(Φ→◇ Ω)
Temporal logic formula

## LPTL
## - automata-theoretical model-checking

State Sequences as Words

- Let AP be the finite set of atomic propositions of the formula f.
- Let $\Sigma = 2^{AP}$ be the alphabet over AP.
- Every sequence of states is an ω word in $\Sigma^\omega$
  - $\alpha = P_0, P_1, P_2, \ldots$ where $P_i = L(s_i)$.
- A word a is a model of formula f iff $\alpha |= f$
- Example: for $f = p \wedge (\neg q \ U \ q)$ $\{p\},\{\},\{q\},\{p,q\}^\omega$
- Let Mod(f) denote the set of models of f.

## LPTL
## - automata-theoretical model-checking

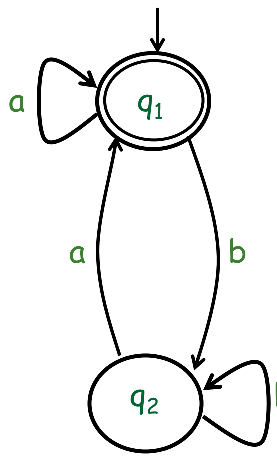
Büchi automaton $A = (Q,\Sigma,\delta,I,F)$

- Q – set of states
- Σ – finite alphabet
- δ – transition relation
- I – set of initial states
- F – set of acceptance states

A run ρ of A on ω word α

$\rho = q_0,q_1,q_2,\ldots$, s.t. $q_0 \in I$ and $(q_i,\alpha_i,q_{i+1}) \in \delta$

ρ is accepting if $Inf(\rho) \cap F \neq \emptyset$



## LPTL
## - automata-theoretical model-checking

**φ:** an LTL fo ith propositions AP.
Construct aton **B(φ)** a cepting
exactly th

work out:
what is ¬(p **U** q) after
pushing the negation ?

egation
**U**q leads to
al blowup!

Naïve co
1. push n
2. simple induction on the

**B(○p)** = ?
**B(**p **U** q**)** = ?
**B(□p)** = ?
**B(**p ∨ q**)** = ?
**B(**p**)** = ?

## LPTL
## - automata-theoretical model-checking

Inductive construction on **φ** :

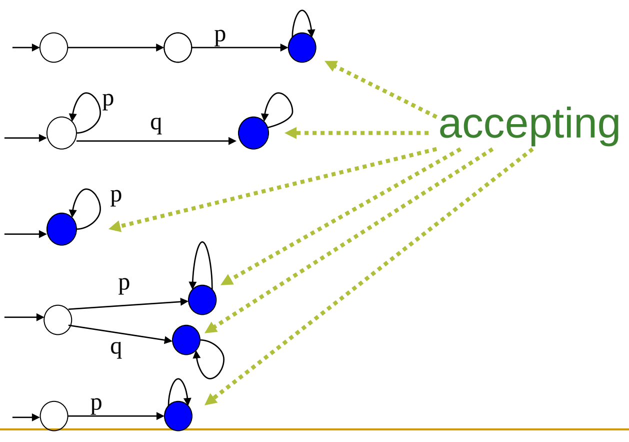
**B(X** p**)** is

**B(**p **U** q**)** is

**B(□** p**)** is

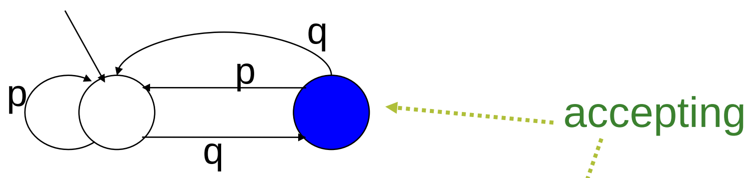
**B(**p ∨ q**)** is

**B(**p**)** is

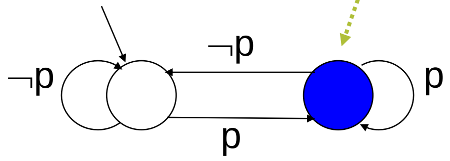


## LPTL - automata-theoretical model-checking

"always p until q": $\Box(pUq)$



"always eventually p": $\Box\Diamond p$



## Workout

Please draw the Buchi automata for the following LTL formulas.

- $(pUq)Ur$
- $\Box((pUq)Ur)$
- $(\Box p)\wedge((pUq)Ur)$
- $(\Diamond p)\vee((qUr)Us)$

## LPTL
## - automata-theoretical model-checking

$\phi$: an LTL formula,

M: a Büchi automata

Model Checking Algorithm $M \models \phi$

- construct $B(\neg\phi)$ for the formula $\phi$
- $M \models \phi$ iff $L(M \times B(\neg\phi)) = \varnothing$

model set of $M \times B(\neg\phi)$

Complexity $O(|M| \times 2^{|\phi|})$

## CTL
## - model-checking

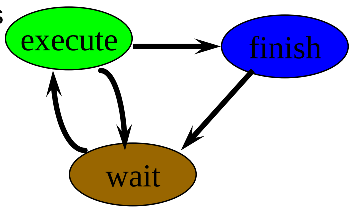
Given a finite Kripke structure M and a CTL formula $\varphi$, is M a model of $\varphi$ ?

- usually, M is a finite-state automata.
- PTIME algorithm.
- When M is generated from a program with variables, its size is easily exponential.

## CTL
## - model-checking algorithm

techniques

- state-space exploration
  - state-spaces represented as finite Kripke structure
    - directed graph
    - nodes: states or possible worlds
    - arcs: state transitions
- regular behaviors



- Usually the state count is astronomical.

## CTL
## - model-checking algorithm (1/6)

Given M and φ,

1. list the subformulas in φ according to their sizes

$$\varphi_0\ \varphi_1\ \varphi_2 \ldots \varphi_n$$

*for all $0 \leq i<j \leq n$ ， $\varphi_j$ is not a subformula of $\varphi_i$*

2. for i=0 to n, label ($\varphi_i$ ) See next page!
3. for all initial states $s_0$ of M, if $\varphi \notin L(s_0)$, return `No!'
4. return `Yes!'

## CTL - model-checking algorithm (2/6)

```
label($\varphi$) {
case p, return;
case $\neg\psi$, for all s, if $\psi \notin L(s)$, $L(s) = L(s) \cup \{\neg\psi\}$
case $\psi_1 \vee \psi_2$, for all s, if $\psi_1 \in L(s)$ or $\psi_2 \in L(s)$,
   $L(s)=L(s)\cup\{\psi_1 \vee \psi_2\}$
case $\exists\bigcirc\psi$, for all s, if $\exists(s,s')$ with $\psi \in L(s')$,
   $L(s)=L(s)\cup\{\exists\bigcirc\psi\}$
case $\exists\psi_1 \mathbf{U} \psi_2$, lfp($\psi_1$, $\psi_2$);
case $\exists\Box\psi$, gfp($\psi$);
}
```

## CTL - model-checking algorithm (3/6)

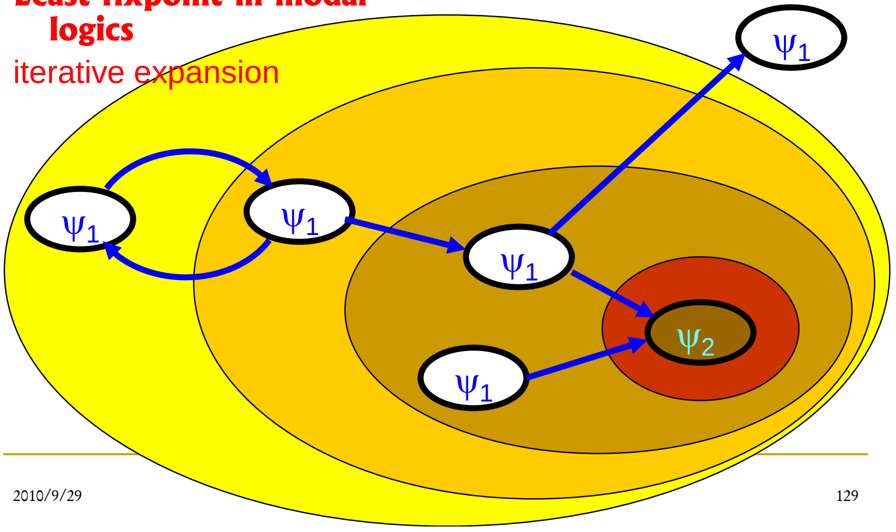
```
lfp($\psi_1$, $\psi_2$) /* least fixpoint algorithm */ {
    for all s, if $\psi_2 \in L(s)$, $L(s)=L(s)\cup\{\exists\psi_1\mathbf{U}\psi_2\}$;
    repeat {
        for all s, if $\psi_1 \in L(s)$ and $\exists(s,s')(\exists\psi_1\mathbf{U}\psi_2 \in L(s'))$,
            $L(s)=L(s)\cup\{\exists\psi_1\mathbf{U}\psi_2\}$;
    } until no more changes to L(s) for any s.
}
```

The procedure terminates since S is finite in the Kripke structure.

## CTL
## - model-checking algorithm (4/6)

**Least fixpoint in modal logics**

iterative expansion



## CTL
## - model-checking algorithm (5/6)

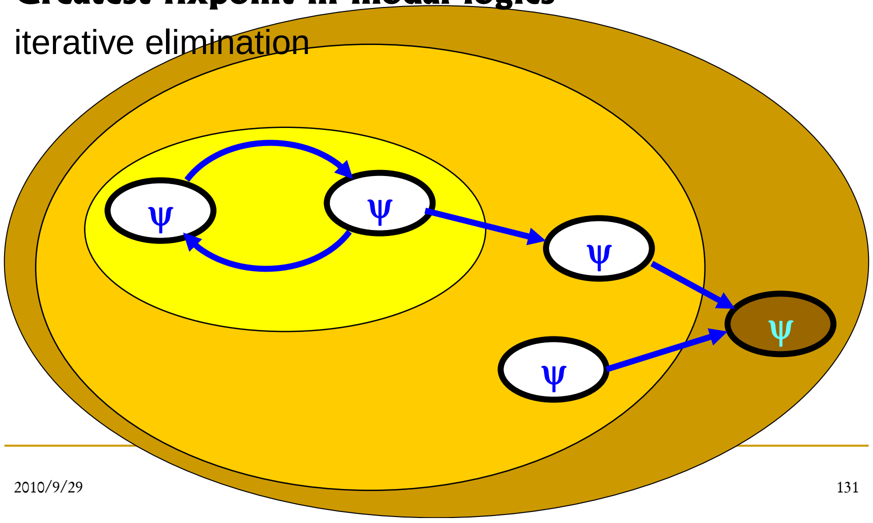
```
gfp(ψ) /* greatest fixpoint algorithm */ {
    for all s, if $\psi \in L(s)$, $L(s)=L(s)\cup\{\exists\Box\psi\}$;
    repeat {
        for all s, if $\exists\Box\psi\in L(s)$ and $\forall(s,s')(\exists\Box\psi\notin L(s'))$,
            $L(s)=L(s) - \{\exists\Box\psi\}$;
    } until no more changes to L(s) for any s.
}
```

The procedure terminates since S is finite in the Kripke structure.

## CTL
## - model-checking algorithm (6/6)

**Greatest fixpoint in modal logics**

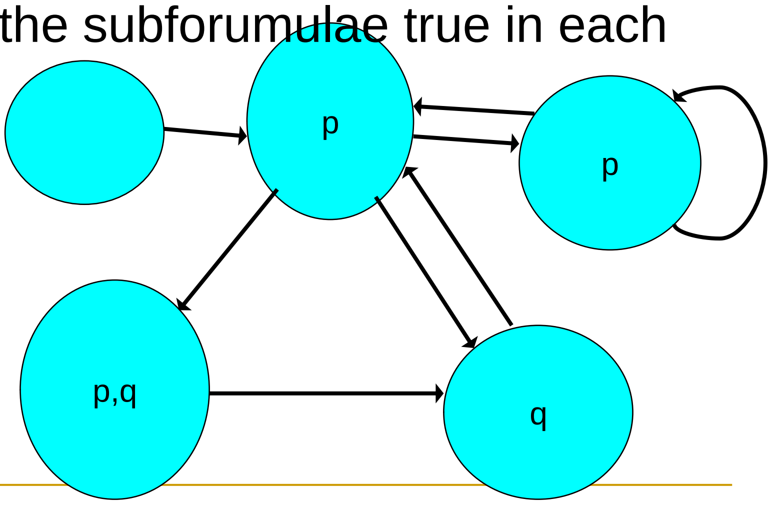
iterative elimination



## (∃○∃p$U$q) ∧ ∃□p

***Labeling funciton:***

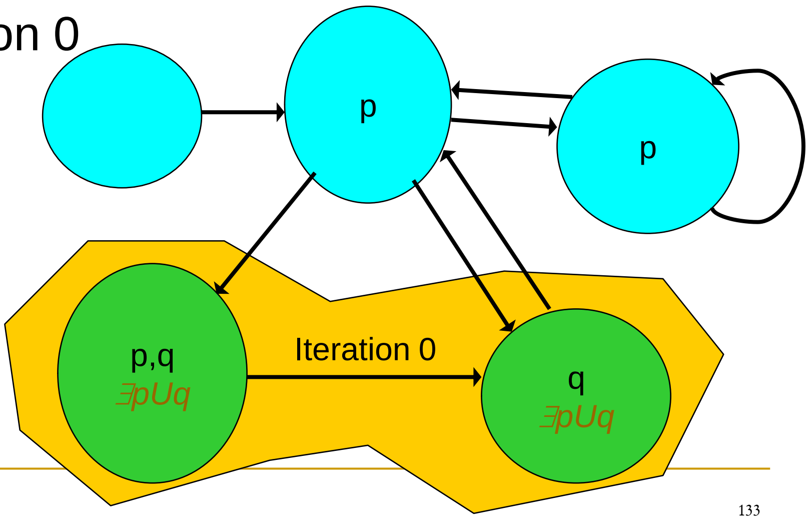
label the subforumulae true in each state.



## (∃○∃p*U*q) ∧ ∃□p

*Evaluating ∃pUq using least fixpoint*

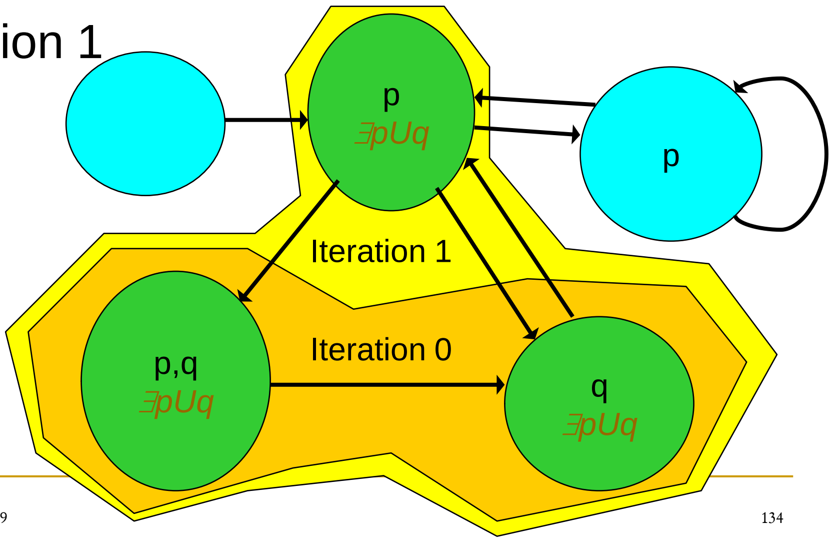
Iteration 0



## (∃○∃p*U*q) ∧ ∃□p

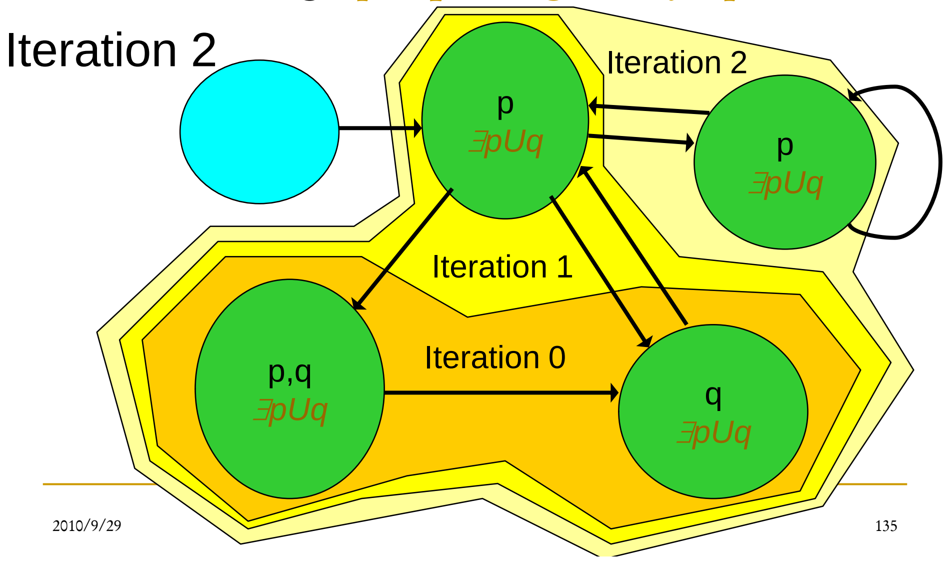
*Evaluating ∃pUq using least fixpoint*

Iteration 1



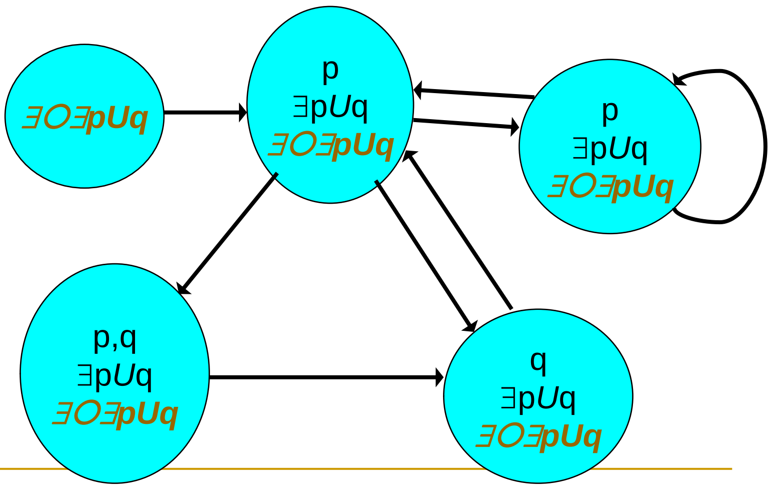
## (∃○∃p*U*q) ∧ ∃□p

*Evaluating ∃pUq using least fixpoint*



## (∃○∃p*U*q) ∧ ∃□p

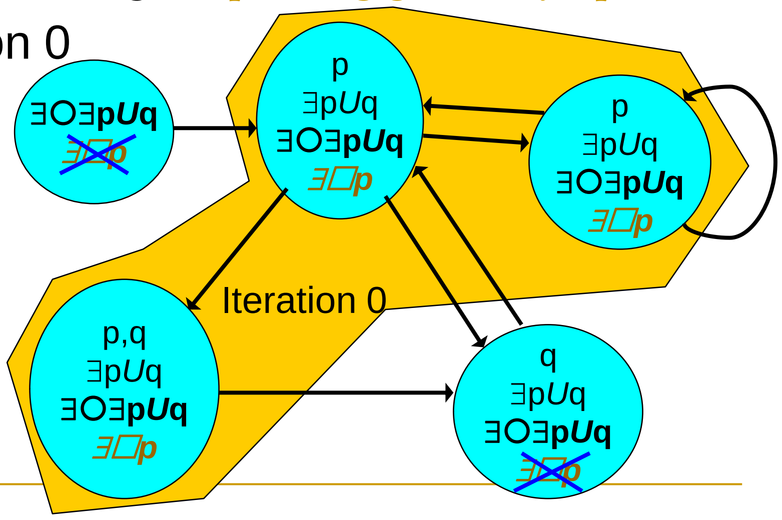
*Evaluating ∃○∃pUq*



## $(\exists\bigcirc\exists p\,\mathcal{U}q) \wedge \exists\Box p$

*Evaluating $\exists\Box p$ using greatest fixpoint*

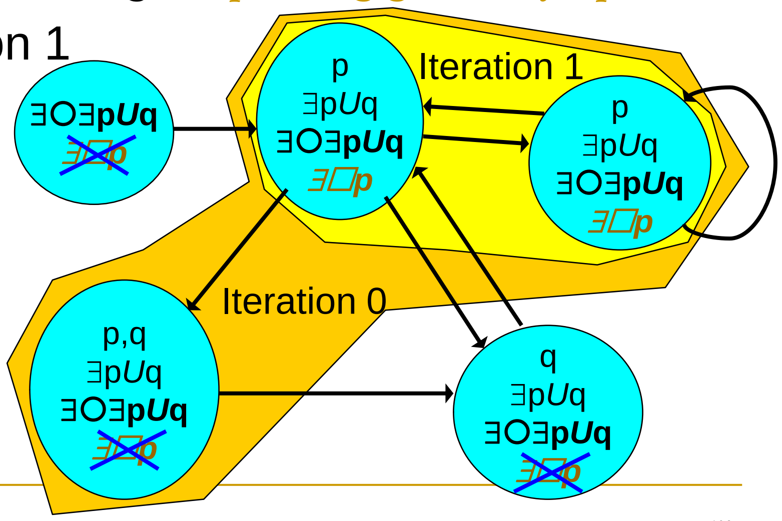
Iteration 0



## $(\exists\bigcirc\exists p\,\mathcal{U}q) \wedge \exists\Box p$

*Evaluating $\exists\Box p$ using greatest fixpoint*

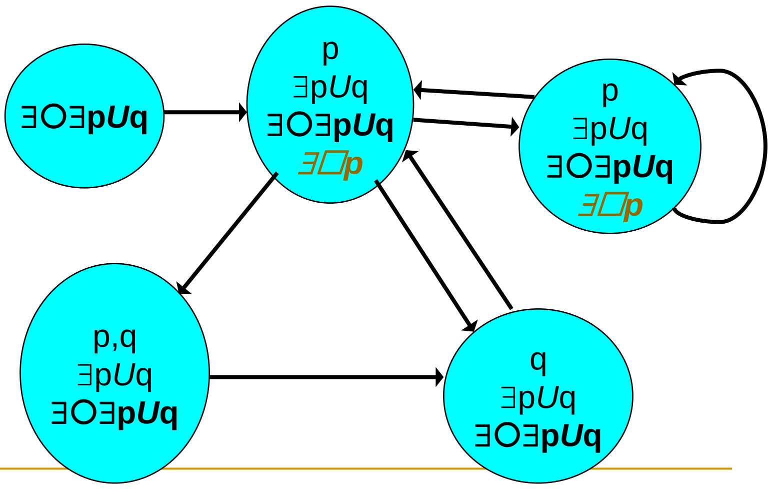
Iteration 1



## (∃○∃p*U*q) ∧ ∃□p

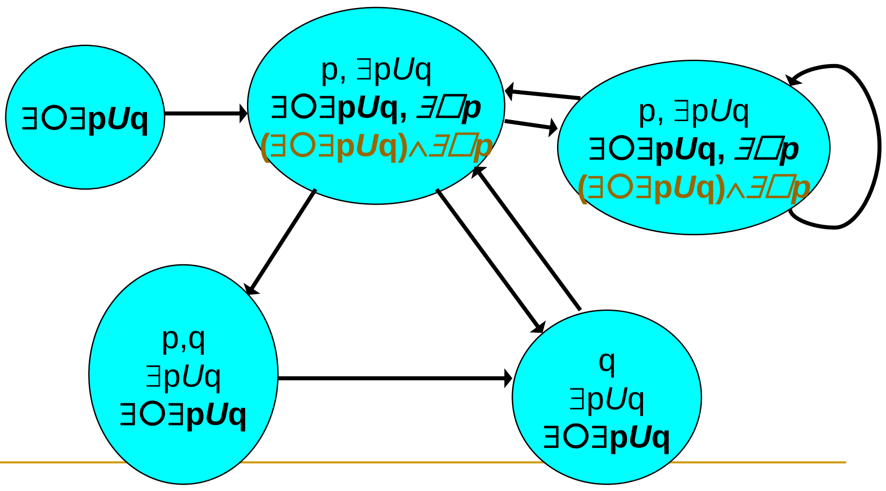
*Evaluating ∃□p using greatest fixpoint*

Result:

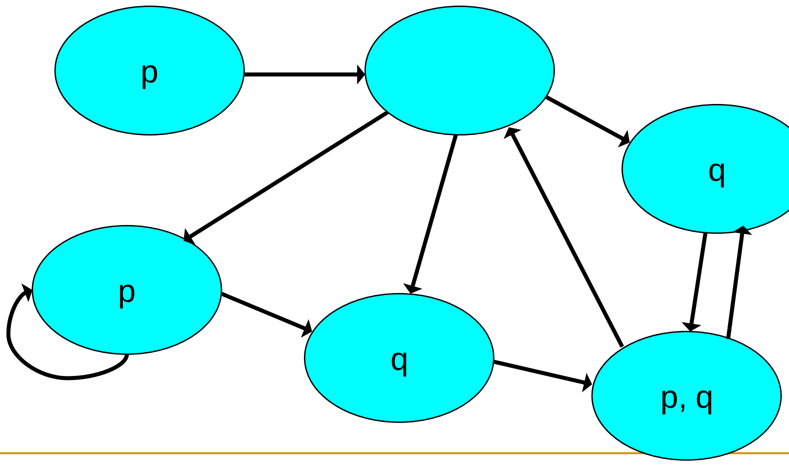


## (∃○∃p*U*q) ∧ ∃□p

*Finally, evaluating (∃○∃p*U*q) ∧ ∃□p*



## Workout: labelling ∃◇(p∧ ∃□q)



## CTL
## - model-checking problem complexities

- The PLTL model-checking problem is PSPACE-complete.
  - definition: Is there a run that satisfies the formula ?
- The PLTL without ○（**modal operator "next"**）model-checking problem is NP-complete.
- The model-checking problem of CTL is PTIME-complete.
- The model-checking problem of CTL* is PSPACE-complete.

## Symbolic until analysis (backward)

$\exists\psi_1 \mathbf{U} \psi_2$

Encode the states with variables $x_0,x_1,\ldots,x_n$.

- the state set as a proposition formula: $S(x_0,x_1,\ldots,x_n)$
- $\psi_1(x_0,x_1,\ldots,x_n)$, $\psi_2(x_0,x_1,\ldots,x_n)$
- the transition set as $R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)$

```
b_0 = $\psi_2(x_0,x_1,\ldots,x_n) \wedge S(x_0,x_1,\ldots,x_n)$; k = 1;
repeat
  $b_k = b_{k-1} \vee \exists x'_0 \exists x'_1 \ldots \exists x'_n( \psi_1(x_0,x_1,\ldots,x_n)$
                              $\wedge R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n)$
                              $\wedge (b_{k-1}\uparrow))$;
  k = k +1;
until $b_k \equiv b_{k-1}$;
```

a least fixpoint procedure

change all umprimed variable in $b_{k-1}$ to primed.

## CTL
## - model-checking algorithm (2/6)

```
slabel($\varphi$ ) {
case p, return $p \wedge S(x_0,x_1,\ldots,x_n)$;
case $\neg\psi$, return $S(x_0,x_1,\ldots,x_n) \wedge \neg$slabel($\psi$);
case $\psi_1 \vee \psi_2$ , return slabel($\psi_1$) $\vee$slabel($\psi_2$)
case $\exists\bigcirc \psi$, return
     $\exists x'_0 \exists x'_1 \ldots \exists x'_n(R(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \wedge$(slabel($\psi$)$\uparrow$));
case $\exists\psi_1 \mathbf{U} \psi_2$, return the symbolic until analysis of
          $\exists$slabel($\psi_1$)U slabel($\psi_2$);
case $\exists\Box \psi$, return the symbolic liveness analysis of
          $\exists\Box$slabel($\psi$);
}
```

## Safety analysis

Given M and p (safety predicate), do all states reachable from initial states in M satisfy p ?

- In model-checking:

  Is M a model of $\forall\Box p$ ?

- Or in risk analysis: Is there a state reachable from initial states in M satisfy p ?

$$\forall\Box p \equiv \neg\exists\Diamond\neg p \equiv \neg\exists \text{true } U \neg p$$

## Reachability analysis

Is there a state s reachable from another state s'?

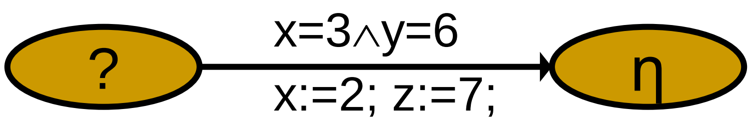
- Encode risk analysis
- Encode the complement of safety analysis
- Most used in real applications

## 2007/06/05 stopped here.

## Symbolic weakest precondition

Assume program with rules

- $x=3 \wedge y=6 \rightarrow x:=2; z:=7;$



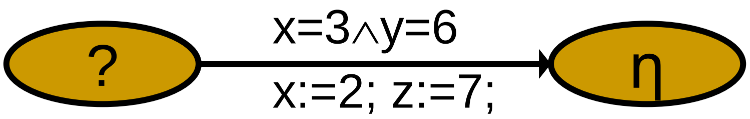
- x, y, z are discrete variables with range declarations

*What is the weakest precondition of* η *for those states before the transitions ?*

## Symbolic weakest precondition

Assume program with rules

- r: $x=3 \wedge y=6 \rightarrow x:=2; z:=7;$



*What is the weakest precondition of* η *for those states before the transitions ?*

$$pre(r, \eta) \overset{\text{def}}{=} x=3 \wedge y=6 \wedge \exists x \exists z(x=2 \wedge z=7 \wedge \eta)$$

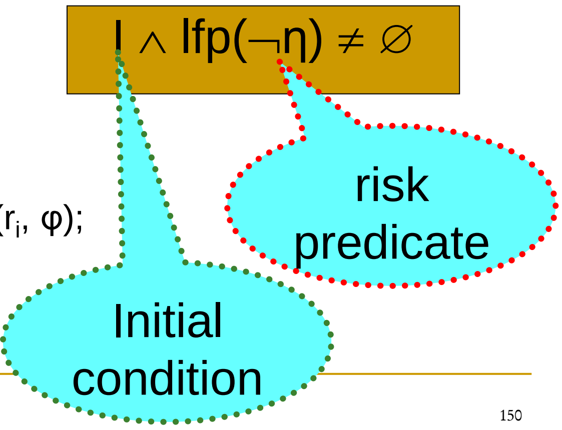
## Symbolic safety analysis

Assume program with rules $r_1, r_2, \ldots, r_n$

*What charcterizes all states that can reach* $\neg\eta$*?*

```
lfp (φ) {
    φ' := false;
    while (φ ≠ φ') {
      φ' := φ ;
      φ := φ ∨ ∨_{i=n} pred(r_i, φ);
    }
    return (φ);
}
```

$\boxed{I \wedge lfp(\neg\eta) \neq \varnothing}$



## Symbolic liveness analysis

Assume program with rules $r_1, r_2, \ldots, r_n$

*What is the charcterization of all states that may not reach* η*?*

```
gfp (φ) {
    φ' := false;
    while (φ ≠ φ') {
      φ' := φ ;
      φ := φ ∧¬∨_{i=n} pred(r_i, φ);
    }
    return (φ);
}
```

$\boxed{I \wedge gfp(\neg\eta) \neq \emptyset}$

Initial condition

negative liveness predicate

## CTL
## - symbolic model-checking with BDD

- System states are described with binary variables.

  $\underline{n \text{ binary variables}} \rightarrow \underline{2^n \text{ states}}$

  $x_1, x_2, \ldots\ldots, x_n$

- we can use a BDD to describe legal states.

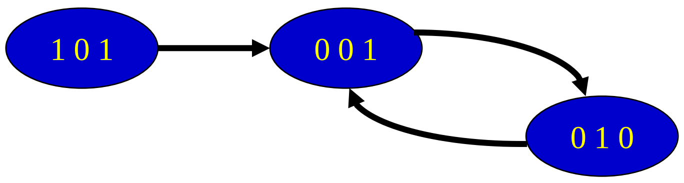
  a Boolean function with *n* binary variables

  **state($x_1, x_2, \ldots\ldots, x_n$)**

## CTL
## - symbolic model-checking with BDD

***Example:***

***$x_1$ $x_2$ $x_3$***



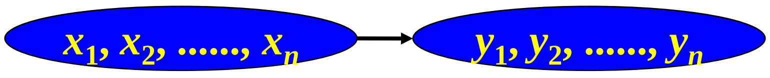
state($x_1, x_2, x_3$) = $(x_1 \wedge \neg x_2 \wedge x_3)$
$\vee$ $(\neg x_1 \wedge \neg x_2 \wedge x_3)$
$\vee$ $(\neg x_1 \wedge x_2 \wedge \neg x_3)$

## CTL
## - symbolic model-checking with BDD

- Transition is a relation between 2 states.
- Thus a relation between 2n binary variables.

  a Boolean function with 2*n* binary variables

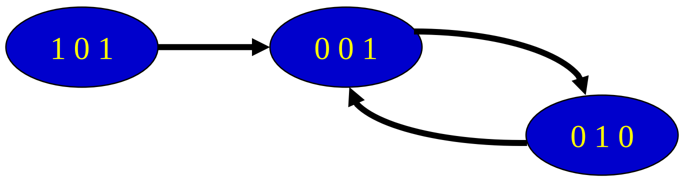
  **transition($x_1, x_2, \ldots\ldots, x_n, y_1, y_2, \ldots\ldots, y_n$)**



## CTL
## - symbolic model-checking with BDD

***Example:***

***$x_1$ $x_2$ $x_3$ $y_1$ $y_2$ $y_3$***



transition$(x_1, x_2, x_3, y_1, y_2, y_3) =$

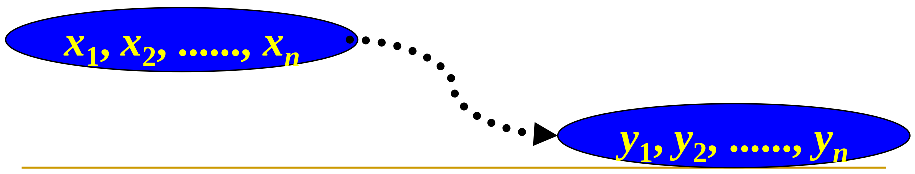
$$(x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge\neg y_2\wedge y_3)$$
$$\vee\quad(\neg x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge y_2\wedge\neg y_3)$$
$$\vee\quad(\neg x_1\wedge x_2\wedge\neg x_3\wedge\neg y_1\wedge\neg y_2\wedge y_3)$$

## CTL
## - symbolic model-checking with BDD

- the reachability relation is also among *2n* binary variables.
- We can use a BDD of *2n* binary variables to describe the reachability relation

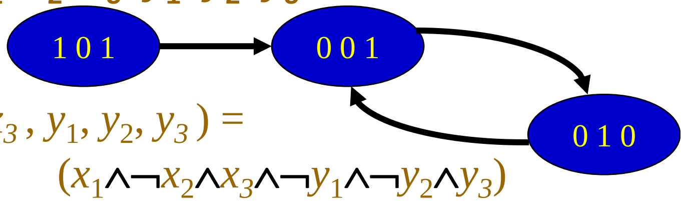
a Boolean funciton of 2*n* bianry variables

**reach($x_1$, $x_2$, ......, $x_n$, $y_1$, $y_2$, ......, $y_n$)**



## CTL
## - symbolic model-checking with BDD

***Example:*** $x_1\ x_2\ x_3\ y_1\ y_2\ y_3$



reach$(x_1, x_2, x_3, y_1, y_2, y_3)$ =

$$
\begin{aligned}
& (x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge\neg y_2\wedge y_3) \\
\vee\quad & (x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge y_2\wedge\neg y_3) \\
\vee\quad & (\neg x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge y_2\wedge\neg y_3) \\
\vee\quad & (\neg x_1\wedge x_2\wedge\neg x_3\wedge\neg y_1\wedge\neg y_2\wedge y_3) \\
\vee\quad & (\neg x_1\wedge\neg x_2\wedge x_3\wedge\neg y_1\wedge\neg y_2\wedge y_3) \\
\vee\quad & (\neg x_1\wedge x_2\wedge\neg x_3\wedge\neg y_1\wedge y_2\wedge\neg y_3)
\end{aligned}
$$

## CTL
## - symbolic model-checking with BDD

**Safety analysis**

**with the BDD for reach($x_1, x_2, \ldots\ldots, x_n, y_1, y_2, \ldots\ldots, y_n$):**

Given initial condition I($x_1, x_2, \ldots\ldots, x_n$) as a BDD and safety condition$\eta$($y_1, y_2, \ldots\ldots, y_n$) as another BDD,
**the system is risky if and only if**
**I$\wedge\neg\eta\wedge$ reach($x_1, x_2, \ldots\ldots, x_n, y_1, y_2, \ldots\ldots, y_n$) is not *false*.**

- Note *true* and *false* both have canonical representations in BDD.

## CTL - symbolic model-checking with BDD

**Reachability analysis**

**with the BDD for reach($x_1, x_2, ......, x_n, y_1, y_2, ......, y_n$):**

Given initial condition I($x_1, x_2, ......, x_n$) as a BDD and goal conditionη($y_1, y_2, ......, y_n$) as another BDD,

**the goal is reachable if and only if**

**I∧η∧ reach($x_1, x_2, ......, x_n, y_1, y_2, ......, y_n$) is not *false*.**

- Note *true* and *false* both have canonical representations in BDD.

## CTL - symbolic model-checking with BDD

Given the BDD of transition ***T*** **($x_1 , x_2 ,....., x_n,y_1,y_2 ,..... ,y_n$),**

***construct the BDD of reach*($x_1, x_2, ......, x_n, y_1, y_2, ......, y_n$)**

- **$B_0$:= *state*($x_1,x_2$ , ...... , $x_n$)∧*T*($x_1,x_2 ,....., x_n,y_1,y_2 ,.....,y_n$)**
- **For *k*:= 1 to ….**

```
        $B_k$ ($x_1,x_2 ,....., x_n,y_1,y_2 ,.....,y_n$)
        := $B_{k-1}$($x_1,x_2 ,....., x_n,y_1,y_2 ,.....,y_n$)
              $\vee\exists z_1..... \exists z_n$ ($B_{k-1}$($x_1,x_2 ,....., x_n,z_1,z_2 ,.....,z_n$)
                         $\wedge B_{k-1}$($z_1,z_2 ,....., z_n,y_1,y_2 ,.....,y_n$))
  until $B_k=B_{k-1}$
```

**$B_k$ ($x_1 ,....., x_n,y_1,.....,y_n$)**

iff the path between the two states is shorter than **$2^k$**

## CTL
## - symbolic model-checking with BDD

For the presentation of the algorithm, we define
***path***$_\psi$**($x_1, x_2, \ldots\ldots, x_n, y_1, y_2, \ldots\ldots, y_n$)**
**instead of *reach*($x_1, x_2, \ldots\ldots, x_n, y_1, y_2, \ldots\ldots, y_n$)**

$$\begin{tikzcd}
\psi \arrow[r] & \psi \arrow[r] & \psi \arrow[r] & \psi \arrow[r] & \cdots \arrow[r] & ?
\end{tikzcd}$$

$(x_1, x_2, \ldots\ldots, x_n)$ $(y_1, y_2, \ldots\ldots, y_n)$

there exists a path from state $(x_1, x_2, \ldots\ldots, x_n)$ to state $(y_1, y_2, \ldots\ldots, y_n)$ along which all states, except the destination, satisfy ψ.

## CTL
## - symbolic model-checking with BDD

- Given a model M and a CTL formulaφ
- the subformulas ofφ:$\varphi_1\varphi_2 \ldots\varphi_n$ in ascending order of sizes

```
For i := 1 to n, do
    if $\varphi_i = x_k$, $B(\varphi_i) := B(x_k) \wedge \mathrm{state}(x_1, x_2, \ldots\ldots, x_n)$
    if $\varphi_i = \psi_1 \vee \psi_2$, $B(\varphi) := B(\varphi_1) \vee B(\varphi_2)$
    if $\varphi_i = \neg\psi$, $B(\varphi_i) := \neg B(\psi)$
    if $\varphi_i = \exists\,\theta\, U\, \psi$,
        $B(\varphi_i) := B(\exists z_1 \ldots\ldots \exists z_n\, path_\theta(x_1,\ldots\ldots,x_n,z_1,\ldots\ldots,z_n) \wedge \psi(z_1,\ldots\ldots,z_n))$
    if $\varphi_i = \exists\Box\psi$,
        $B(\varphi_i) := B(\exists z_1 \ldots\ldots \exists z_n\, path_\psi(x_1,\ldots\ldots,x_n, z_1,\ldots\ldots,z_n)$
                        $\wedge\, path_\psi(z_1,\ldots\ldots,z_n, z_1,\ldots\ldots,z_n))$
                  $:= B(\exists z_1 \ldots\ldots \exists z_n\, \exists w_1 \ldots\ldots \exists w_n$
                        $path_\psi(x_1,\ldots\ldots,x_n, z_1,\ldots\ldots,z_n)$
                        $\wedge\, path_\psi(z_1,\ldots\ldots,z_n, w_1,\ldots\ldots,w_n)$
                        $\wedge \bigwedge_{1\le i\le n} (z_i = 0 \wedge w_i = 0 \vee z_i = 1 \wedge w_i = 1))$
```

## CTL
## - symbolic model-checking with BDD

Construct the BDD of $\exists z_1 \ldots\ldots \exists z_n\, B(z_1, \ldots\ldots, z_n)$?

- $\exists z_n\, B(z_1, \ldots\ldots, z_n) = B(z_1, \ldots\ldots, z_{n-1}, 0) \vee B(z_1, \ldots\ldots, z_{n-1}, 1)$
  $= (z_n{=}0 \wedge B(z_1, \ldots\ldots, z_{n-1}, z_n)) \vee (z_n{=}1 \wedge B(z_1, \ldots\ldots, z_{n-1}, z_n))$
- For ***i*** := ***n*-1** to **1**, do

  $\exists z_i \ldots\ldots \exists z_n\, B(z_1, \ldots\ldots, z_n)$

  $= (\exists z_{i+1} \ldots\ldots \exists z_n\, B(z_1, \ldots\ldots, z_{i-1}, 0, z_{i+1}, \ldots\ldots, z_n))$

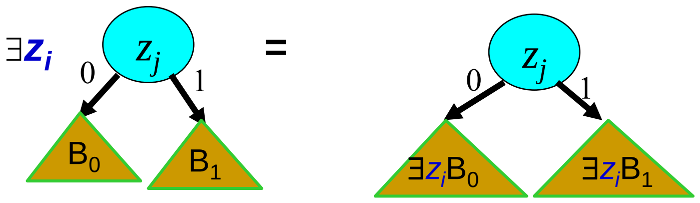
  $\vee (\exists z_{i+1} \ldots\ldots \exists z_n\, B(z_1, \ldots\ldots, z_{i-1}, 1, z_{i+1}, \ldots\ldots, z_n))$

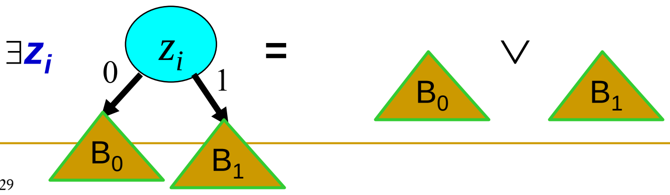
## CTL
## - symbolic model-checking with BDD

**Transition BDD: T ($x_1, \ldots\ldots, x_n, y_1, \ldots\ldots, y_n$) and CTL formula $\varphi$**
**the subformula of $\varphi$: $\varphi_1 \varphi_2 \ldots \varphi_n$** in ascending order of sizes

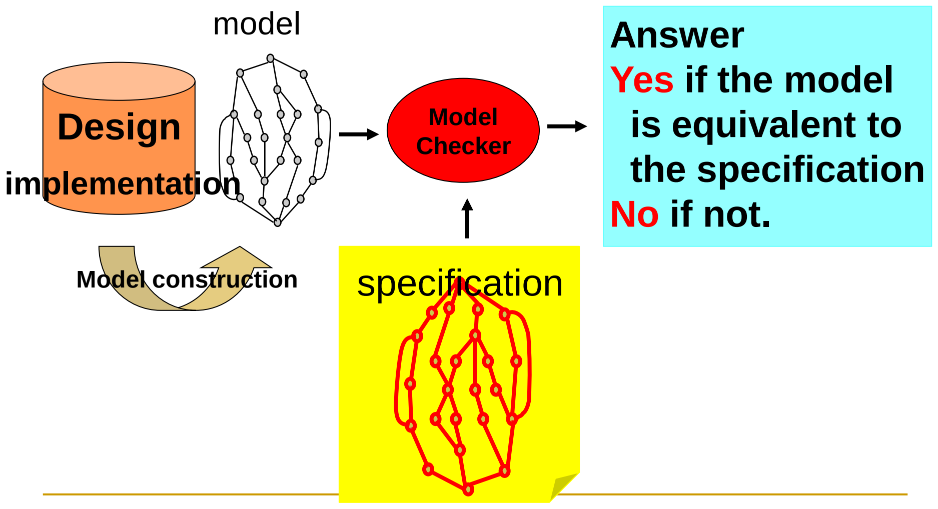
```
For i := 1 to n, do
  if $\varphi_i = x_k$, $B(\varphi_i) := B(x_k) \wedge state(x_1, x_2, \ldots\ldots, x_n)$
  if $\varphi_i = \psi_1 \vee \psi_2$, $B(\varphi_i) := B(\psi_1) \vee B(\psi_2)$
  if $\varphi_i = \neg\psi_1$, $B(\varphi_i) := \neg B(\psi_1) \wedge state(x_1, x_2, \ldots\ldots, x_n)$
  if $\varphi_i = \exists\bigcirc\psi_1$, $B(\varphi_i) := \exists y_1 \ldots\ldots \exists y_n(T(x_1, \ldots\ldots, x_n, y_1, \ldots\ldots, y_n)$
                                   $\wedge rename(B(\psi_1), x_1 \to y_1, \ldots, x_n \to y_n))$
  if $\varphi_i = \exists\psi_1 \mathrm{U}\ \psi_2$,
     $B(\varphi_i) := lfp\ Z.(B(\psi_2) \vee \exists y_1 \ldots\ldots \exists y_n($ $T(x_1, \ldots\ldots, x_n, y_1, \ldots\ldots, y_n)$
                                                $\wedge B(\psi_1)$
                                                $\wedge rename(Z, x_1 \to y_1, \ldots, x_n \to y_n)$
                              )                 )
  if $\varphi_i = \exists\Box\psi_1$,
     $B(\varphi_i) := gfp\ Z.(B(\psi_1) \wedge \exists y_1 \ldots\ldots \exists y_n($ $T(x_1, \ldots\ldots, x_n, y_1, \ldots\ldots, y_n)$
                                           $\wedge rename(Z, x_1 \to y_1, \ldots\ldots, x_n \to y_n)$
                              )                )
```

## Implementation of $\exists z_i\, B(z_1, \ldots\ldots, z_n)$

- $\exists z_i \boxed{0} = \boxed{0}$; $\exists z_i \boxed{1} = \boxed{1}$;
- 

- 

## Bisimulation Framework

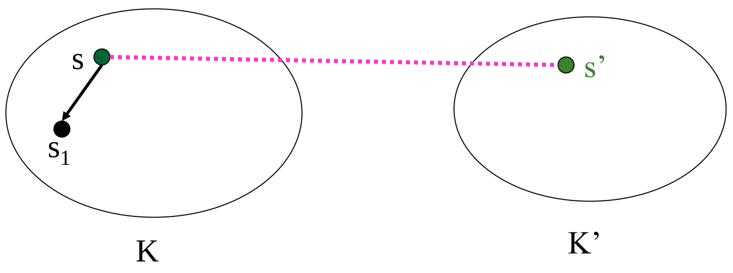


**Answer**
**Yes if the model is equivalent to the specification**
**No if not.**

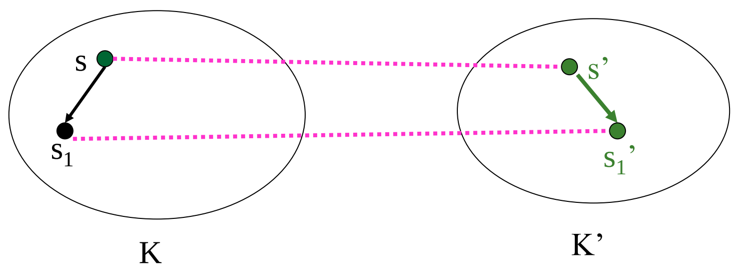
## Bisimulation-checking

- $K = (S, S_0, R, AP, L)$
  $K' = (S', S_0', R', AP, L')$
- Note K and K' use the same set of atomic propositions AP.
- $B \in S \times S'$ is a bisimulation relation between K and K' iff for every B(s, s'):
  - L(s) = L'(s') (BSIM 1)
  - If $R(s, s_1)$, then there exists $s_1'$ such that $R'(s', s_1')$ and $B(s_1, s_1')$. (BISIM 2)
  - If $R(s', s_2')$, then there exists $s_2$ such that $R(s, s_2)$ and $B(s_2, s_2')$. (BISIM 3)

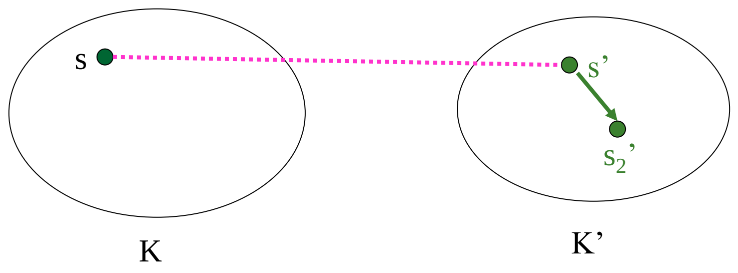
## Bisimulations



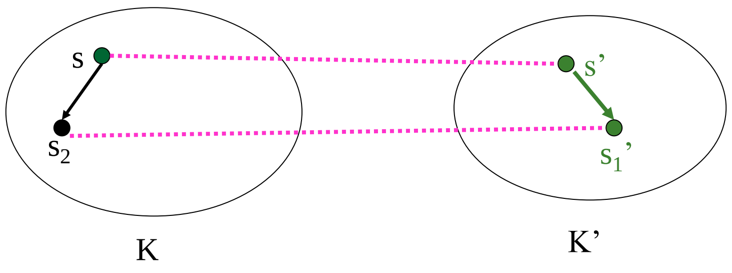
## Bisimulations



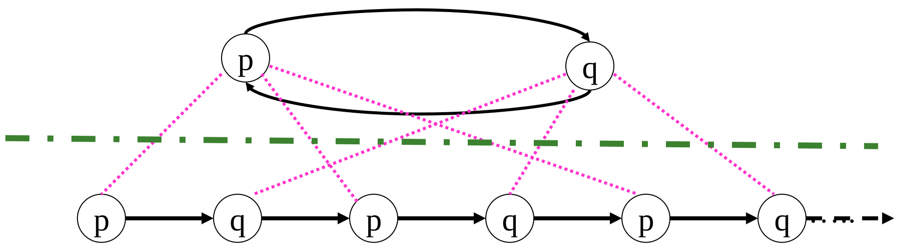
## Bisimulations



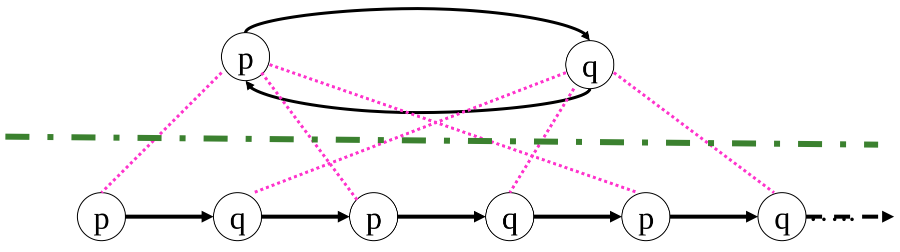
## Bisimulations



## Examples

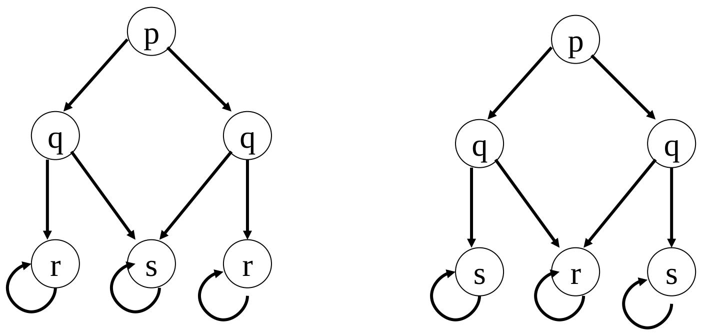


## Examples

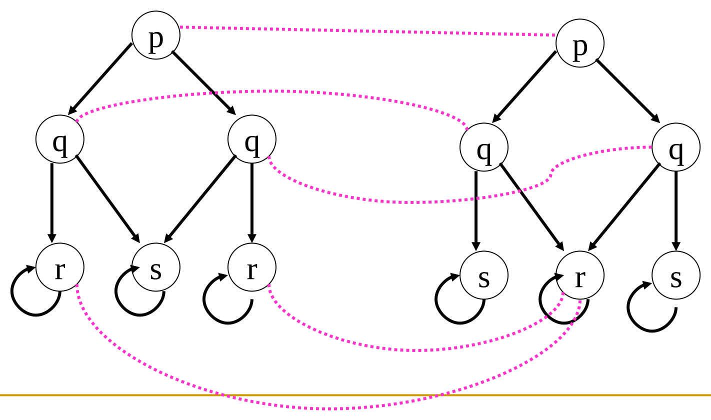


Unwinding preserves bisimulation

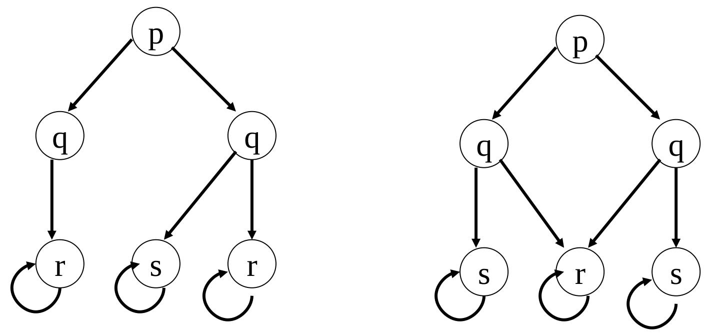
## Examples



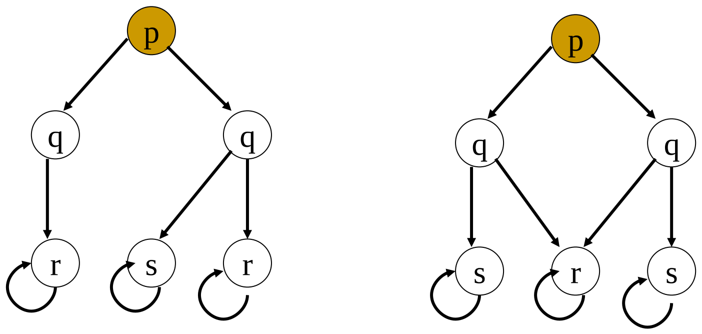
## Examples



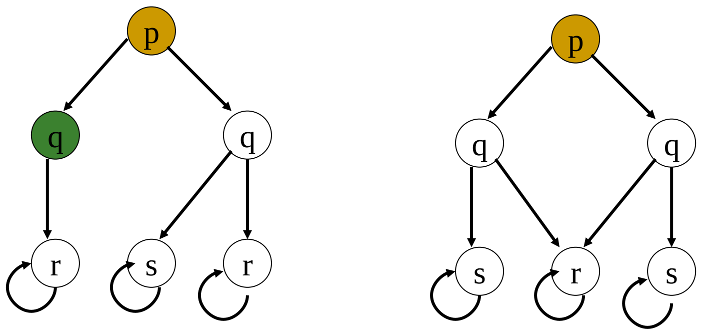
## Examples



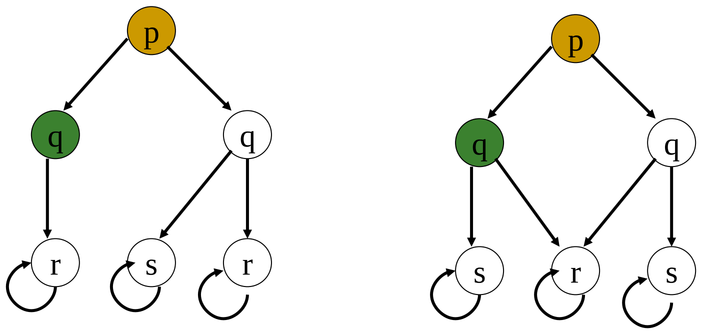
## Examples



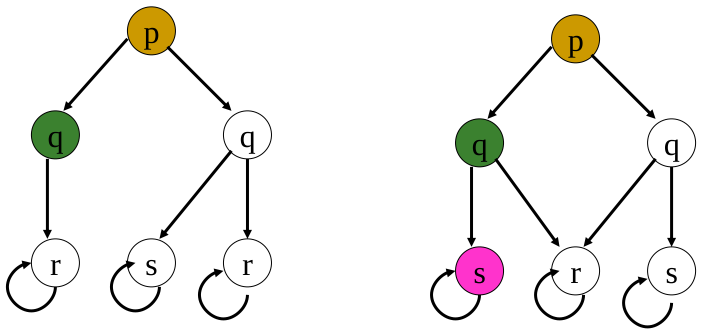
## Examples



## Examples



## Examples



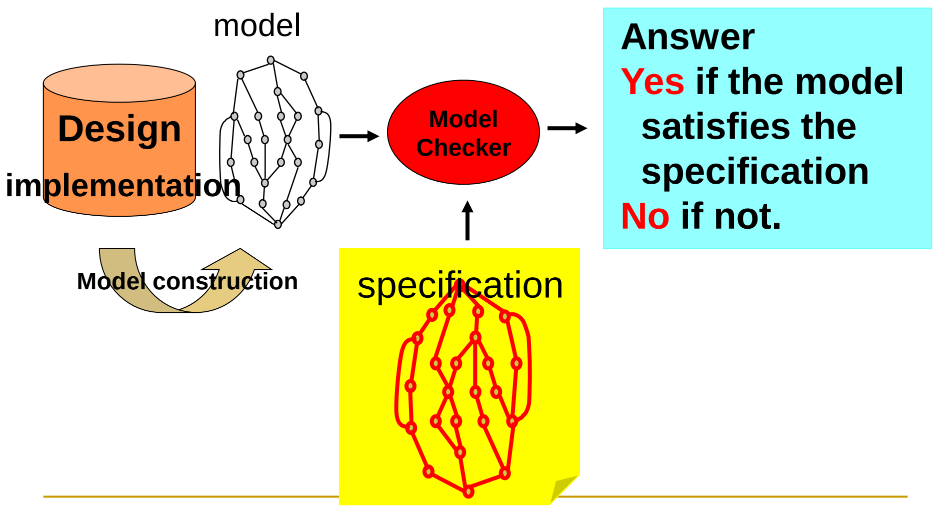
## Bisimulations

- K = (S, $S_0$, R, AP, L)
- K'= (S', $S_0$', R', AP, L')
- K and K' are bisimilar (bisimulation equivalent) iff there exists a bisimulation relation B µ S £ S' between K and K' such that:
  - For each $s_0$ in $S_0$ there exists $s_0$' in $S_0$' such that B($s_0$ , $s_0$').
  - For each $s_0$' in $S_0$' there exists $s_0$ in $S_0$ such that B($s_0$ , $s_0$').

## The Preservation Property.

- K = (S, $S_0$, R, AP, L)
  K'= (S', $S_0$', R', AP, L')
- B µ S £ S', a bisimulation.
- Suppose B(s, s').
- FACT: For any CTL formula $\psi$ (over AP), K, s ² $\psi$ iff K', s' ² $\psi$.
- If K' is smaller than K this is worth something.

## Simulation Framework



## Simulation-checking

- K = (S, $S_0$, R, AP, L)
  K’= (S’, $S_0$’, R’, AP, L’)
- Note K and K’ use the same set of atomic propositions AP.
- B µ S £ S’ is a simulation relation between K and K’ iff for every B(s, s’):
  - L(s) = L’(s’) (BSIM 1)
  - If R(s, $s_1$), then there exists $s_1$’ such that R’(s’, $s_1$’) and B($s_1$, $s_1$’). (BISIM 2)

## Simulations

- K = (S, $S_0$, R, AP, L)
- K'= (S', $S_0$', R', AP, L')
- K is simulated by (implements or refines) K' iff there exists a simulation relation B µ S £ S' between K and K' such that for each $s_0$ in $S_0$ there exists $s_0$' in $S_0$' such that B($s_0$ , $s_0$').

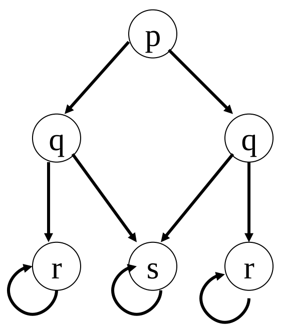
## Simulation Quotients

- K = (S, $S_0$, R, AP, L)
- There is a maximal simulation B µ S £ S.
  - Let R be this bisimulation.
  - [s] = {s' j s R s'}.
- R can be computed "easily".
- K' = K / R is the bisimulation quotient of K.

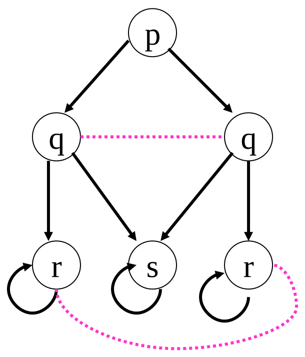
## Bisimulation Quotient

- K = (S, $S_0$, R, AP, L)
- [s] = {s' j s R s'}.
- K' = K / R = (S', $S'_0$, R', AP,L').
  - S' = {[s] j s 2 S}
  - $S'_0$ = {[$s_0$] j $s_0$ 2 $S_0$}
  - R' = {([s], [s']) j R($s_1$, $s_1$') for some $s_1$ 2 [s] and $s_1$' 2 [s']}
  - L'([s]) = L(s).

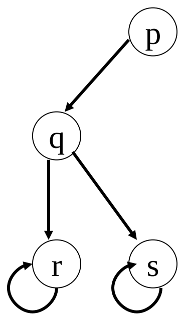
## Examples



## Examples



## Examples



## Abstractions

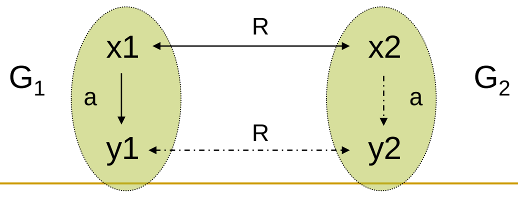
- Bisimulations don't produce often large reduction.
- Try notions such as simulations, data abstractions, symmetry reductions, partial order reductions etc.
- Not all properties may be preserved.
- They may not be preserved in a strong sense.

## Graph Simulation

**Definition** Two edge-labeled graphs $G_1$, $G_2$

A *simulation* is a relation R between nodes:

- if $(x_1, x_2) \in R$, and $(x_1,a,y_1) \in G_1$, then exists $(x_2,a,y_2) \in G_2$ (same label) s.t. $(y_1,y_2) \in R$



Note: if we insist that R be a function ➔ graph homeomorphism

## Graph Bisimulation

**Definition** Two edge-labeled graphs G1, G2

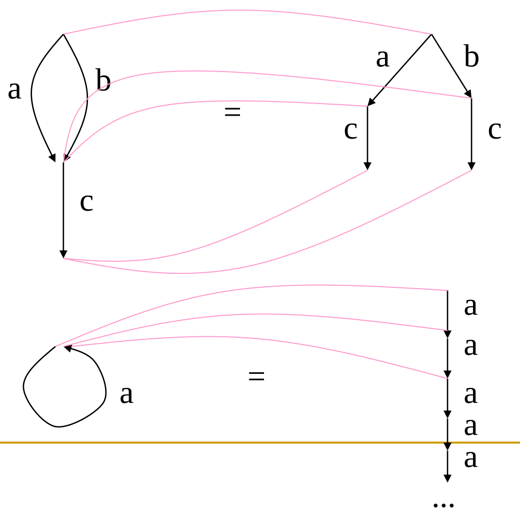
A *bisimulation* is a relation R between nodes s.t. both R and $R^{-1}$ are simulations

## Set Semantics for Semistructured Data

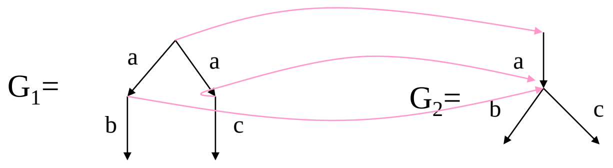
**Definition** Two rooted graphs $G_1$, $G_2$ are equal if there exists a bisimulation R from $G_1$ to $G_2$ such that (root($G_1$), root($G_2$)) $\in$ R

- Notation: $G_1 \approx G_2$
- For trees, this is precisely our earlier definition

## Examples of Bisimilar Graphs



## Examples of non-Bisimilar Graphs



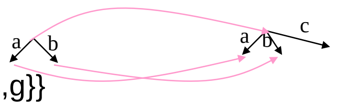
- This is a *simulation* but not a *bisimulation*
  - Why ?
- Notice: $G_1$, $G_2$ have the same sets of *paths*

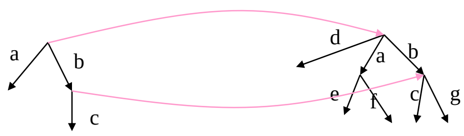
## Examples of Simulation

- Simulation acts like “subset”

  {a, b} $\subseteq$ {a, b, c}

  {a, b:{c}} $\subseteq$ {d, a:{e,f}, b:{c,g}}



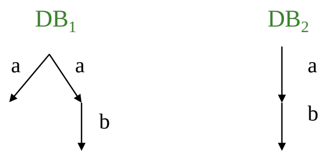


- Question:
- if DB$_1$ $\subseteq$ DB$_2$ and DB$_2$ $\subseteq$ DB$_1$ then DB$_1$ $\approx$ DB$_2$ ?

## Answer

if DB$_1$ $\subseteq$ DB$_2$ and DB$_2$ $\subseteq$ DB$_1$ then DB$_1$ $\approx$ DB$_2$ ?

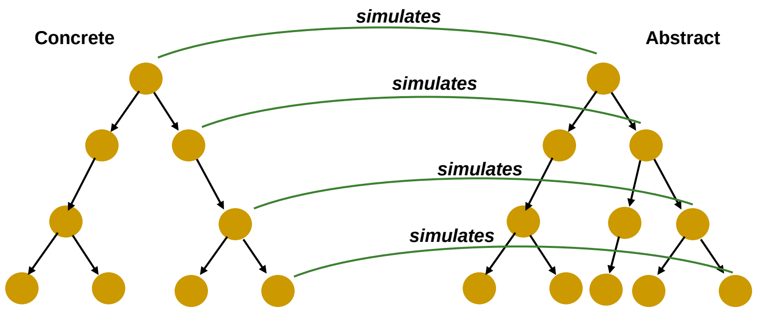
No. Here is a counter example:



DB$_1$ $\subseteq$ DB$_2$ and DB$_2$ $\subseteq$ DB$_1$ but NOT DB$_1$ $\approx$ DB$_2$

## Path Simulation

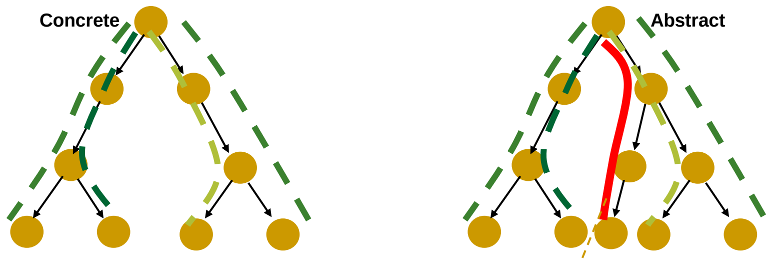
Intuition: every path in concrete system is simulated by a path in abstract system



A concrete path $s_1, s_2, \ldots$ is simulated by an abstract path $a_1, a_2, \ldots$ if $Sim(s_i, a_i)$ for all i.

## Computation Simulation

Intuition: every path in concrete system is simulated by a path in abstract system

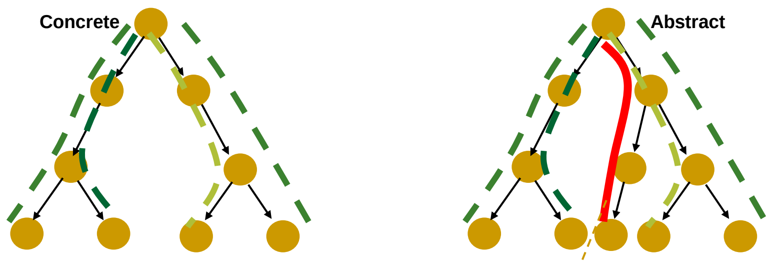


*Infeasible path due to over-approximation.*

There may be extra paths (termed “infeasible” paths) that are not present in the concrete system. These are due to the approximate nature of our computation with abstract tokens. Specifically, they arise from the over-approximations in test branching discussed previously.

## Reflection of LTL Properties

If there is a violating path in the concrete system, then there is a violating path in the abstract system, since the simulation property guarantees that each concrete trace has a corresponding trace in the abstract system. Technically, this means that properties are *reflected* by abstraction.



*Infeasible path due to over-approximation.*

If there is a violating path in the abstract system, then *there is not necessarily* a violating path in the concrete system, since the violating abstract trace may be an infeasible path due to over-approximation. Technically, this means that properties are not *preserved* by abstraction.

## Facts About a (Bi)Simulation

- The empty set is always a (bi)simulation
- If R, R' are (bi)simulations, so is R U R'
- Hence, there always exists a *maximal* (bi)simulation:
  - Checking if $DB_1=DB_2$: compute the maximal bisimulation R, then test $(root(DB_1),root(DB_2))$ in R

## Computing a (Bi)Simulation

- Computing the maximal (bi)simulation:
  - Start with R = nodes($G_1$) x nodes($G_2$)
  - While exists ($x_1$, $x_2$) $\in$ R that violates the definition, remove ($x_1$, $x_2$) from R
- This runs in polynomial time ! Better:
  - O((m+n)log(m+n)) for bisimulation
  - O(m n) for simulation
  - Compare to finding a graph homeomorphism !

$\longrightarrow$ NP Complete